

Diploma Thesis

Semi-Automatic Mapping of Structured
Data to Visual Variables

submitted by

Jan Polowinski

born 03.03.1981 in Gelsenkirchen

Technische Universität Dresden

Fakultät Informatik
Institut für Software- und Multimediatechnik
Lehrstuhl Softwaretechnologie

Supervisor: Dr. rer. nat. habil. Uwe Aßmann
Professor: Dr. rer. nat. habil. Uwe Aßmann

Submitted August 25, 2007

Confirmation

I confirm that I independently prepared the thesis and that I used only the references and auxiliary means indicated in the thesis.

Dresden, August 25, 2007

Table of Contents

Abstract	x
1. Introduction	1
2. Visualization of Structured Data in General	4
2.1. Global and Local Interfaces	4
2.2. Steps of the Visualization Process	4
2.3. Existing Visual Selection Mechanisms	6
2.4. Existing Visualizations of Structured Data	12
2.5. Categorizing SemVis	25
3. Requirements for a Flexible Visualization	27
3.1. Actors	27
3.2. Use Cases	27
4. Fresnel, a Standard Display Vocabulary for RDF	31
4.1. Fresnel Lenses	31
4.2. Fresnel Formats	33
4.3. Fresnel Groups	33
4.4. Primaries (Starting Points)	33
4.5. Selectors and Inference	34
4.6. Application and Reusability	34
4.7. Implementation	35
5. A Visualization Ontology	37
5.1. Describing and Formalizing the Field of Visualization	37
5.2. Overview	37
5.3. VisualVariable	38
5.4. DiscreteVisualValue	39
5.5. VisualElement	41
5.6. VisualizationStructure	42
5.7. VisualizationPlatform	42
5.8. PresentationScenario	43
5.9. Facts	44
6. A Novel Mapping Vocabulary for Semantic Visualization	45
6.1. Overview	45
6.2. Mapping	46
6.3. PropertyMapping	47
6.4. ImplicitMapping	48
6.5. ExplicitMapping	53
6.6. MixedMapping	54
6.7. ComplexMapping	55
6.8. Inference	58
6.9. Explicit Display of Relations	58
6.10. Limitations	59
7. A Model-Driven Architecture for Flexible Visualization	60
7.1. A Model-Driven Architecture	61
7.2. Applications of the MDA Pattern	62
7.3. Complete System Overview	71
7.4. Additional Knowledge of the System	72
7.5. Comparison to the Graphical Modelling Framework — GMF	77
8. Visualization Platforms	80
8.1. Extensible 3D (X3D)	80
8.2. Scalable Vector Graphics (SVG)	81
8.3. XHTML + CSS	82

8.4. Text	82
9. Outlook and Conclusion	84
9.1. Advanced Mapping Vocabulary	84
9.2. Reusing Standardized Ontologies	84
9.3. Enabling Dynamic, Interaction and Animation	84
9.4. Implementation and Evaluation	85
9.5. Conclusion	85
Glossary	86
Bibliography	87
A.	90
A.1. Schemata	90

List of Figures

2.1. Selection Step (Graph to Subgraph)	4
2.2. Structuring Step (Graph to Ordered Tree)	5
2.3. Formatting Step (Visual Information Added)	6
2.4. Graph Display of a Query in GLOO	7
2.5. Option List for the Facet <i>fieldOfInterest (Multiple Values Selected)</i>	9
2.6. Slider Widget for the Facet <i>size</i>	9
2.7. Text Input Widget for the Facet <i>price (Range Selectable)</i>	10
2.8. Theoretical Set Widget <i>for Purely Nominal Data</i>	10
2.9. Tree Widget for the Facet <i>loves (Sorted by the Property hates)</i>	11
2.10. Option List for the Facet <i>importance (Single Value Selected)</i>	12
2.11. Option List for the Facet <i>importance (Quantitative Information)</i>	12
2.12. <i>GraphViz</i> Graph Visualization (Module Dependencies)	13
2.13. <i>Walrus</i> Graph Visualization (Networking)	14
2.14. <i>IsaViz</i> — FOAF data	15
2.15. Relation Browser — Geographical Data	16
2.16. FLAMENCO - Data on Nobel Prize Winners	17
2.17. Longwell — Primary Selection View	18
2.18. Longwell — Browsing View	19
2.19. <i>mSpace</i>	20
2.20. <i>mSpace</i> (Multiple Columns per Facet)	20
2.21. <i>FacetMap</i>	21
2.22. <i>Ebay Express</i> — Search for a Hammock	22
2.23. <i>Amazon</i> — Books for Children over 10	23
2.24. <i>Google Base</i> — Search for Cars	24
2.25. <i>Mambo</i> — Aspect Browsing of Music Data By Title	25
3.1. Use Cases — Extension by the Programmer	28
3.2. Use Cases — Configuration by the Admin	28
3.3. Activity Diagram — Configuration by the Admin	29
3.4. Use Cases — Browsing of the User	30
4.1. Use of Sublenses in <i>Fresnel</i>	32
4.2. <i>Fresnel</i> Result Tree	35
5.1. The <i>Graphics.owl</i> ontology — Overview	38
5.2. The <i>Graphics.owl</i> Ontology — <i>VisualElements</i>	41
6.1. Subclasses of Mapping	45
6.2. <i>Mapping.owl</i>	46
6.3. Mappings between Discrete and Continuous Values	49
6.4. Mapping Ordinal Data (Tree Structure)	49
6.5. Explicit Mapping of Discrete Unstructured Values	53
6.6. <i>ComplexMapping</i>	56
7.1. Variability of <i>SemVis</i>	60
7.2. <i>SemVis</i> as a Model-Driven Architecture	61
7.3. Illustration of the <i>SemVis</i> Model as Java Objects	66
7.4. X3D Instantiation of the Abstract <i>SemVis</i> Box Model	71
7.5. <i>SemVis</i> Complete Architecture	72
7.6. Overview of all Ontologies and Definition Files	73
7.7. Architecture of the Graphical Modelling Framework	77
7.8. Architecture of <i>SemVis</i>	78
8.1. Example of X3D Output — US Presidents	81
8.2. Example of X3D Output — Events of History	81
8.3. Example of SVG Output — US Presidents	82

8.4. Example of PDF Output — Events of History 83

List of Tables

2.1. Possibilities of Selecting a Value Range	9
2.2. Possibilities of Selecting a Value Range	11
2.3. Longwell — SIMILE-Timeline View	19
9.1. Comparison of Dynamic Possibilities between X3D, Java3D and SVG	84

List of Examples

4.1. Fresnel Lens Definition	32
4.2. Fresnel Format Definition	33
4.3. Fresnel Group Definition	33
4.4. Definition of Primaries	34
4.5. Inference in Fresnel	34
5.1. VisualizationPlatform Class	43
5.2. PresentationScenario Class	44
6.1. Ontology Mapping by Inference	58
7.1. RDF Description of the Domain Model	63
7.2. Example Fresnel Definition	64
7.3. Fresnel Tree as XML Output	65
7.4. Mapping of the Facet party to the Visual Variable Color	67
7.5. SemVis Model as XML Output	68
7.6. Generated X3D Code	69
7.7. Generated X3D Code (continued)	70
7.8. Definition of XHTML and X3D as VisualizationPlatforms	74
7.9. Definition of Presentation Scenarios	75
7.10. Definition of a Visualization Structure	75
7.11. User Limitations of the Filtering	76
7.12. User Limitations of the Mapping	76

Abstract

While semantic web data is machine understandable and well suited for advanced filtering, in its raw representation it is not conveniently understandable to humans. Therefore, visualization is needed. A core challenge when visualizing the structured but heterogeneous data, turned out to be a flexible mapping to Visual Variables. This work deals with a highly flexible, semi-automatic solution with a maximum support of the visualization process, reducing the mapping possibilities to a useful subset. The basis for this is knowledge, concerning metrics and structure of the data on the one hand and available visualization structures, platforms and common graphical facts on the other hand — provided by a novel basic visualization ontology. A declarative, platform-independent mapping vocabulary and a framework was developed, utilizing current standards from the semantic web and the Model-Driven Architecture (MDA).

Chapter 1. Introduction

Besides the information that is stored as texts on websites that can only be retrieved by string comparison through search engines, there is an increasing amount of structured data in the world wide web. By the use of standardized description languages as the Resource Description Framework [RDF] and the Web Ontology Language [OWL] this data can be connected as part of the *semantic web* [BHL01] to enable new services.

This work proposes a possibility of quickly generating visualizations for arbitrary structured data that has been transformed to RDF, including data sources not known as yet. Fields for visualization are: Semantic web data, product data in web shops and auctions, *Wikipedia* data (especially from a semantic version), tagged data ([Flickr], [Del.icio.us], etc.), *Personal Information Management* (PIM) data about e-mails, appointments, documents, bookmarks and contacts, [Gnowsis].

Besides the life sciences, which were pioneers in the creation of semantic web content [Sam07], there are two fields that have the potential to offer large sets of semantic web data: Semantic wikis and shopping systems, which will be further discussed in the following two sections.

Semantic wikis try to benefit from the simple way of authoring wikis and use the wiki principle to offer an easy way to collect structured data. The authoring of RDF data could not, until now, be easily done by domain experts, but only by knowledge representation experts. The use of semantic wikis could change that, and therefore the data collected by them could become a major source of semantic web data.

The *Semantic MediaWiki* software [Semantic MediaWiki] is an extension to the *Media Wiki* [MediaWiki], used for the *Wikipedia* for example. It employs categories and typed links pointing to other instances or to attributes for simple datatypes, each defined by an article of its own. The agreement on the precise meaning of a relation or category can be achieved via the community process, since each reference and class has its page. The *Semantic Media Wiki* software offers the possibility of exporting its data as RDF which can be used as input to SemVis. This is demonstrated by some of the example data, which has been gathered from the wiki of *Ontoworld*¹, a wiki for the semantic web community.

Other semantic wikis exist, such as *SemperWiki*². Many more have been developed in the context of Personal Information Management. For example, *Gnowsis*, a *Semantic Desktop Environment* published by the *Knowledge Management Lab* of the DFKI (Deutsche Forschungszentrum für Künstliche Intelligenz) integrates a Semantic Wiki, [Gnowsis].

Ebay, *Amazon* and especially *Google Base* administrate huge amounts of structured product data, that becomes more and more interconnected and fine grained. E.g. relationships like *Similar Products* are established and detailed product attributes like *Size*, *Resolution*, *Energy Consumption* for a display device, or *Size*, *Color*, *Material*, *Sex* for a textile are stored. Every product needs different attributes, only some of them can be shared by generalized products. This allows for very convenient product search by limiting each characteristic to a certain desired value or value range and for convenient comparison of products, but results in very heterogeneous data. Data, as the semantic web technologies can handle it however.

¹<http://wiki.ontoworld.org/wiki>

²<http://www.semperwiki.org>

It is a characteristic of semantic web data, that it does not inherit any representation. Data and representation are intentionally cleanly separated and hence semantic web data always needs to be transformed and visualized, to be shown to humans. Because of the variety of possible domains it is worthwhile offering a generic framework for visualization.

This work is based on the experiences made with a former work on the topic *Visualization of Large Data Sets In 3D-Space* [Pol06]. This former work had some problems which we want to list here, in order to explain our motivation for building a new system.

So far the mapping behavior in the framework was hard coded into the source code. This tight mapping of data to Visual Variables and models is clearly undesirable and had to be resolved. Equally the selection of data to be displayed was hard coded and the data (i.e. RDF data and ontologies) was accessed directly via the API of an RDF repository³ instead of using a query language. Furthermore the old framework was limited to a specific platform, [Java3D], as the only possible output platform. A fourth major issue was the lack of any support for the definition of filtering and mapping by the system.

The new system, described in this work, is referenced to as *SemVis* as a working title. *SemVis* tries to accomplish three main goals in order to overcome the problems of the old framework. These goals are an exchangeable and reusable definition of presentation knowledge, a maximum of variability by variable platforms for the presentation and a semi-automatism of the visualization with as much support for the user as possible.

The exchangeable, loosely coupled definition of the visualization is achieved by the use of declarative display and mapping languages. The definition of this presentation knowledge may, to some great extend, be reused in other visualization systems, since standards are used wherever possible. Equally, the data repository can more easily be exchanged, because the selection of data is done using the query language [SPARQL].

While the old framework was limited to the *Java3D* platform, the *Model-Driven Architecture* [MDA] allows for a description of the presentation knowledge independently of the final visualization platform. From this description, it can generate code for several output formats including *XHTML*, *X3D* and *SVG*. The user can choose from a variety of platforms that are described to *SemVis*.

SemVis does not try to offer a fully automatic visualization of arbitrary data in a domain-independent way. In case of a fully automatic approach, as necessary for browsers for unknown data, the visualization has to be reduced to a least common denominator, which offers little value. Based on the idea that only the user⁴ knows the presentation goals, the new framework introduces a semi-automatic approach. This requires adding presentation knowledge, but results in specialized, custom-tailored visualizations. *SemVis* tries to support the user as much as possible with a pre-reduced set of possibilities and suggests default values based on metrics and general graphical knowledge. To enable this kind of interaction with the user, the system requires a graphical interface.

This work is structured as follows: Chapter 2 offers a general overview of visualization of semantic web data to be able to categorize *SemVis* and compare it to existing work from the field of graph visualization and semantic web browsers. Chapter 3 defines requirements for a flexible visualization system to give an overview of the activities of the different actors in *SemVis*, without going into detail. The following three chapters introduce necessary vocabularies for these processes. After *Fresnel* as a standard

³The system currently uses *Open RDFs Sesame* as an RDF repository [SESAME].

⁴The term user is used in a general way. The different actors involved are described in detail in Section 3.1. Here the admin of a system is meant, who is familiar with the domain.

display vocabulary for RDF is introduced, the newly created *Graphics.owl* ontology (Chapter 5) and the *Mapping.owl* vocabulary (Chapter 6) are explained. Chapter 7 is about the system architecture of SemVis, putting it into the context of the *Model-Driven Architecture* and explaining the transformation steps and required knowledge of the system. Chapter 8 shortly describes a selection of possible output formats for SemVis. Finally Chapter 9 summarizes the improvements as well as the remaining limitations and resulting next steps that have to be taken.

Chapter 2. Visualization of Structured Data in General

This chapter is about the general challenges when visualizing structured data. To be able to categorize SemVis into existing solutions for the visualization of structured data, characteristics of visualization systems are introduced in Section 2.1 and the overall visualization process is subdivided into subsequent steps in Section 2.2. Afterwards existing techniques for the visualization of the filtering process by the user (Section 2.3) and for the visualization of the resulting filtered data (Section 2.4) are presented and compared using the terms we introduced before. Section 2.5 finally puts SemVis into the existing categories and compares it to other solutions.

2.1. Global and Local Interfaces

According to [Rut05], the visualizations of structured data, especially RDF data, can be subdivided into those with global interfaces, those with local interfaces and those which integrate both into one solution.

Global interfaces focus on the overview and the structure of the data. The graph visualization presented in Section 2.4.1 are a good example for this.

Local interfaces concentrate on details of a particular object. The objects of interest are linked to each other from instance to instance. This way the RDF data becomes browseable. The popular OWL editor [PROTÉGÉ] is a characteristic example of this category.

A third group of visualizations provides integrated interfaces, which seamlessly connect the overview with the instance view by marking the current location in the global interface and offering orientation in the semantic space.

2.2. Steps of the Visualization Process

Based on the work of [Rut05] we consider the visualization process of semantic web data is three fold, consisting of *selection*, *structuring* and a *formatting* step. The SemVis mapping vocabulary contributes to the *formatting* part, however techniques concerning selection, such as faceted browsing and structuring techniques such as a mechanism to pick a specific view on the RDF data are used by the framework as well. Each of these steps is described in the following subsections.

2.2.1. Searching and Filtering (Selection)

Searching and filtering in the field of computing can be seen as the act of narrowing down an initial set of items. Restricting the values for facets of the data then leads to a *selection* of instances.

Visual possibilities for the selection process are described in Section 2.3 in detail.

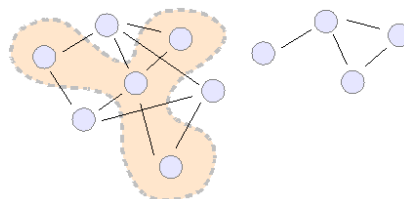


Figure 2.1. Selection Step (Graph to Subgraph)

2.2.. Picking a View Point (Structuring)

The structured data, when available as RDF, represents a directed graph that must not even be acyclic or connected and can be seen from a variety of view points. As an example consider a knowledge base containing events and persons that are involved in those events. One view could take the persons primarily and show the related events for each of the persons. Another view could put the events in the center of interest and subordinate the persons involved.

What is an advantage when searching and filtering turns out to be unuseful when displaying the data in a common way. Representations of knowledge like texts and diagrams need data that has hierarchy and sequence (i.e. an ordered tree), because this is the way we are used to structure our information from text books, [Rut05]. So the next step, when visualizing RDF data, is to structure the selected data. This can be reduced to the problem of extracting an ordered tree.

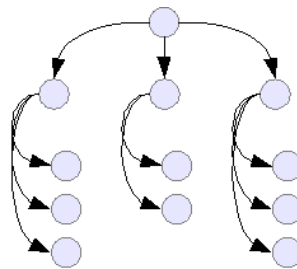


Figure 2.2. Structuring Step (Graph to Ordered Tree)

While most commonly represented as XML documents, RDF data can be represented in diverse notations such as [N3]. But even the RDF/XML serialization, which definitely is a tree by the nature of XML, can not be used for visualization purposes, since this serialization is arbitrary. It can have many shapes while still saying nothing about the way the information is intended to be presented [Wal03]. What needs to be done, is to add presentation knowledge to pick one of the view points on the data and extract a meaningful tree, representing this view point.

Many solutions have been developed to structure RDF data. Due to the lack of a RDF/XML canonicalization, described above, it is not possible to process RDF/XML with plain XSLT¹. *Xenon* and *RDF Twig* overcame this problem with extensions to XSLT. However, these were procedural approaches, which were limited to a specific output format and did not support a standard query language such as SPARQL.

Declarative approaches for the description of display information have also been developed. However, they still suffered from being display paradigm specific, for example the *Graph Style Sheets* (GSS) used in the *IsaViz* Browser [IsaViz] and the *Haystack Slide ontology*, [QHK03], [BLP05].

The solution that is chosen in SemVis for the purpose of structuring is the *Fresnel RDF display vocabulary*², which is a fully declarative, domain- and paradigm-agnostic approach (detailed description in Chapter 4). Fresnel goes further than the structuring part and can also be used to define the formatting part, which will be described in the next section.

¹XSLT — XSL Transformations: A language for transformations between XML documents (<http://www.w3.org/TR/xslt20/>)

²More precisely the *Fresnel Lens Vocabulary* is responsible for the description of the structuring process.

2.2.3. Defining Visual Appearance (Formatting)

The formatting process can be subdivided into a styling part and a mapping part. The styling part, the decoration of elements, is done with the *Fresnel Format Vocabulary*, which is described in detail in Chapter 4. The mapping part, the assignment of facets to Visual Variables, is described in detail in Chapter 6.

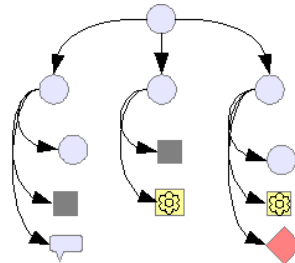


Figure 2.3. Formatting Step (Visual Information Added)

2.3. Existing Visual Selection Mechanisms

Visualization aspects primarily apply to the presentation of the selected data, but already the selection of the data and the configuration of the visualization itself can be done in a visual way, supported by the system. The visualization of the dynamic, incremental selection mechanism is not the main concern of this work. Hence, it shall only briefly be mentioned which user interfaces for this visual filtering exist.

When a user wants to filter the data he could use a query language to do so. SPARQL is such a language, that offers a convenient means to formulate queries over RDF data. However, there remain two problematic issues with formulating SPARQL queries. First, the user has to know the vocabulary of the query language, what can be tolerated, secondly, to make restrictions on values, the user has to know at least part of the data she wants to query in advance, which is usually not the case. Therefore the user has to be guided by the system somehow when filtering the data.

There are two approaches to achieve that, which are described below: The visual construction of queries (Section 2.3.1) and the *Faceted³ Browsing* paradigm (Section 2.3.2). They both have in common that neither the vocabulary nor the data needs to be known and syntax errors are impossible through the user interface. Common to both approaches is the fact that they incrementally refine the set of selected instances, always offering the possibility to broaden the selection again by clearing restrictions. The use of a GUI has the additional advantage of being able to preset probable, frequently used values, though this is not used by any of the studied mechanisms so far.

The last section (Section 2.3.3) proposes user interface widgets for different selection tasks, depending on the type of a facet.

2.3.1. Visual Query Construction

Instead of writing in a query language, the formulation of a query string can be visually constructed with the help of systems, using a *Visual Query Languages* (VQL) such as GLOO, the *Graphical Query Language for OWL Ontologies* [FH06] or the visual query system developed in the context of SEWASIE⁴ [CM03]. Similar efforts have been made

⁴Semantic Webs and AgentS in Integrated Economies

in the field of databases, eliminating the need for an domain expert to learn a language such as SQL before being able to search a database. The domain independence of the RDF query language is retained by the visual language.

GLOO maps to *nRQL*⁵ and represents classes, individuals and also operators such as *NOT*, *OR* and *AND* as geometrical objects. These objects can be connected by named arrows, representing properties, to form a tree as a visual representation of the query. The operators allow for building complex queries by combining simple ones. Besides the advantages that count for both query construction systems, GLOO offers information on result cardinalities to give feedback to the user. Figure 2.4 shows a simple example for a graph representing a query in GLOO.

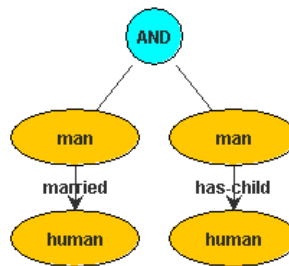


Figure 2.4. Graph Display of a Query in GLOO

The SEWASIE visual query system uses an iterative refinement process and proposes the technique of *intensional navigation* to build conjunctive queries. By this it is very close to the Faceted Browsers that will be described in the following section. In addition to the graphical display of the query as a tree structure, the query is steadily output in natural language, where the tree structure corresponds to nested sentences. This way the user can check the semantical correctness of her selections and navigate to a subquery by clicking on a part of the generated sentence. It should be noted that only the generation of natural language is supported (and necessary), not the parsing. The system employs additional ontologies to enrich the searched data, e.g. by providing extra labels for associations.

2.3.2. Faceted Browsing

Faceted Browsing performs the incremental refinement of a set of results by restricting values of the data's facets. In contrast to visual queries languages, the faceted browsing does not return the constructed query explicitly, but, after each refinement, directly presents the user the reduced set of results and the possibilities to further choose from.

A characteristic, common to most *Facet Browsers* is the fact that the possibilities the user has to choose from are restricted to the possibilities which do not lead to an empty result set. This way it is guaranteed, that a search always returns at least one instance. Likewise the cardinality of the results from a potential restriction is mostly displayed for each facet value as a preview.

Some Facet Browsers perform an automatic facet ranking to allow the user to make restrictions in an appropriate order. This automatic facet ranking can be based on metrics as discussed by [ODD06]. Oren et.al. distinguish between *Descriptors*, the facets that are well suitable for describing an item, and *Navigators* which are suitable for navigating through the data. The quality of a facet as a *Navigator* can be calculated by three

⁵A query language offered by the OWL-DL reasoner Racer

measurable properties: *Predicate Balance*, *Object Cardinality* and *Predicate Frequency*.

Since Faceted Browsers are a visualization of the selection mechanism as well as a means to visualize the data itself, examples for several Faceted Browsers are described in the data representation section (c.f. Section 2.4.2).

Starting Points (Primaries)

The definition of starting points is essential for the user to explore the data with a Faceted Browser. In contrast to an explicit query, an explorative interaction needs such a base to further specialize.

Classes that are used as starting points are referred to as *Primaries* in this work to be consistent with existing vocabulary (see also Section 4.4). They were called *Entities* in former work [Pol06] and are also referred to as *Focal Points* in [Rut05].

The choice of starting points is highly dependant on the domain and the intended purpose of the browsing. Good Primaries in the domain of history data are, for instance, the classes *Event*, *Person*, *Century*, *Location* or *Field of Interest*. Bad Primaries for the field of history, because they are only of peripheral interest or too specialized, are classes such as *Continent*, *Invention*, *Art Style*, *Weapon*, etc. However, these classes *might* be useful as Primaries in another context.

2.3.3. Selection Widgets

While filtering, suitable widgets to choose the facet values have to be offered. We propose a selection of these widgets based on the characteristics of the facet to be filtered. Most of the current Faceted Browsers, introduced later on, use the same widgets for all facet types, for example, and not much attention has been paid to the fact that the facets inherit a structure, which can be helpfully reflected by the widget.

Entering Facet Values

A first distinction of selection widgets can be made according to their possibilities for entering the desired facet values. This section lists criteria concerning the quality of widgets for entering the desired facet values. In the next section criteria concerning the display of facet characteristics are given.

Range Selection

Choosing a single value is often undesirable. The user often rather wants to define a vague range of values as it is usually the case with facets such as *time* and *price*. Special widgets, defining a minimum and a maximum value can be used in this case.

Sliders with two markers or two separate *text input fields* are possible means to define ranges. Also *option lists* can show the selection of a range by marking all values within a maximum and a minimum value.

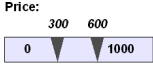
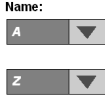

		
<p>Slider Widget for the Facet <i>price</i> (Range Selectable)</p>	<p>Option List Widget for the Facet <i>name</i> (Range Selectable)</p>	<p>Option List for the Facet <i>importance</i> (Range Selected)</p>

Table 2.1. Possibilities of Selecting a Value Range

Multiple Selection

An alternative to this, is the possibility of selecting multiple discrete values of a facet. However, this has its limitations for facets with a large number of values and is not possible at all for continuous values. Multiple selection is possible, for example, in *option lists*, lists of *checkboxes* and could also be implemented for *tree maps*.

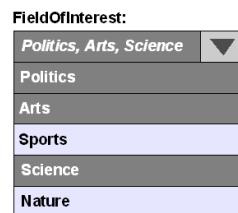


Figure 2.5. Option List for the Facet fieldOfInterest (Multiple Values Selected)

Selection of Continuous Values

A further criterion is the ability to define continuous values. This can be performed by a text input field, selection wheels or by sliders (c.f. Figure 2.6).

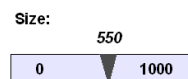


Figure 2.6. Slider Widget for the Facet size

Free Input

A last distinctive feature is the free input of values (c.f. Figure 2.7) in contrast to the selection of predefined values. The free text input of *ObjectProperty* values (by the identifier) requires recall, therefore free text input is better suited to the selection of numeral values.

Price from ... to ...

Figure 2.7. Text Input Widget for the Facet price (Range Selectable)

Reflecting the Facets Characteristics

The following criteria concern the widgets ability to reflect the characteristics of the facet, in especially the relationship between the facet values.

In the field of statistics, values are categorized into *nominal*, *ordinal* and *quantitative* data representing different scales of information on the relationship between the facet values. Depending on the category to which the facet values belong to, different widgets are appropriate to reflect this information.

Nominal Data

If there is no relationship between the facet values, except for the fact that they are all different and can be distinguished by their name, the data is (only) nominal. A trivial sorting however, can be performed using the alphanumerical order of the values titles. An example for selection widgets for nominal data is given in Figure 2.5. An *option list*, however, already suggests the existence of an order, even if there is none. An appropriate widget for a set of values without any order should actually look like Figure 2.8, stressing the unrelatedness of the values. But normally it does not matter if the values are ordered anyway. Often the alphanumerical sorting can be chosen to give meaning to the order.

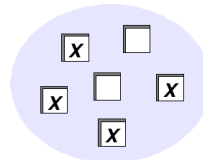


Figure 2.8. Theoretical Set Widget for Purely Nominal Data

Ordinal Data

If there is an order relation between the values, the data can be referred to as ordinal. The relationship between the facet values that is reflected by the widget, does not necessarily have to be the facet property itself. It might also be another relation that fulfils the condition of not leaving the facets range. Consequently the range and the domain of the relation have to be the same and have to equal the facets range (c.f. Figure 2.9). The special case of a widget that is structured by the facet itself is shown in Table 2.2 (left side).

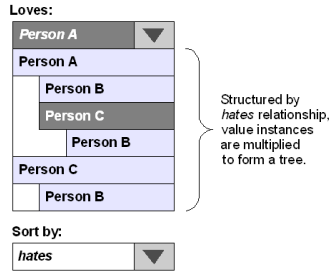


Figure 2.9. Tree Widget for the Facet loves (Sorted by the Property hates)

The topology build by the order relation might be a *sequence*, a *tree* or a DAG. An example for a facet that has a tree structure is the facet *takesPlaceIn* (domain *Event*). It has the class *GeographicalRegion* as a range. For the moment, assume that *GeographicalRegions* are related to each other via an exclusive *partOf* relationship⁶. Facets that have a hierarchical topology can be well displayed by the classical directory tree (already seen in Figure 2.9) or by a *tree map* widget [JS91], [SCM+06] as shown in Table 2.2 (right side). It can further be distinguished, if the selection can be performed with one click, because the whole hierarchy is explicitly shown in an expanded state, or the selection can be performed by an incremental refinement, level by level.

<p>Tree Widget for the Geographical Facet <i>locatedIn</i></p>	<p>Tree Map Widget for the Geographical Facet <i>locatedIn</i></p>

Table 2.2. Possibilities of Selecting a Value Range

Other hierarchical facets include *isA* or *partOf* relationships, i.e. component relationships of composed products and body parts.

An example for a facet that is structured by a strict order connecting its discrete values is the facet *hasImportance*. An appropriate widget could be a simple option list as shown in Figure 2.10.

⁶This is a simplification. The relation between geographical regions is actually not an exclusive *partOf* relationship, but there are countries being part of more than one region. This results in a DAG, that can however be transformed into a tree by the duplication of child elements (also c.f. Figure 2.9).

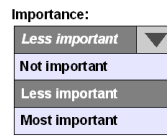


Figure 2.10. Option List for the Facet importance (Single Value Selected)

Quantitative Data

Quantitative data provides information on the ratio of values or at least on the ratio of differences between pairs of values (depending on the fact if the *Null* was chosen in a meaningful way, or not). This quantitative information can be reflected by the visual variable *Area* for example.

A *Tree Map* is able to provide quantitative information by the proportions of areas and also a *Pie Chart* could be used as a widget. The *Option List Widget* can also be extended to represent quantitative information by using different sizes for each value to reflect the ratio to other values (c.f. Figure 2.11).

Quantitative data always provides sequence, yet the values are not discrete, but continuous. So it is always possible to handle only the ordinal data aspect. All numeral simple data types such as *xsd:float* and *xsd:integer* offer quantitative data.

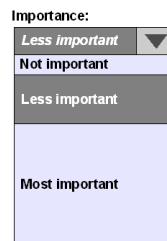


Figure 2.11. Option List for the Facet importance (Quantitative Information)

2.4. Existing Visualizations of Structured Data

The following two sections describe existing visualization techniques subdivided into the fields Graph Visualization and Faceted Browsing, always considering the principles mentioned before.

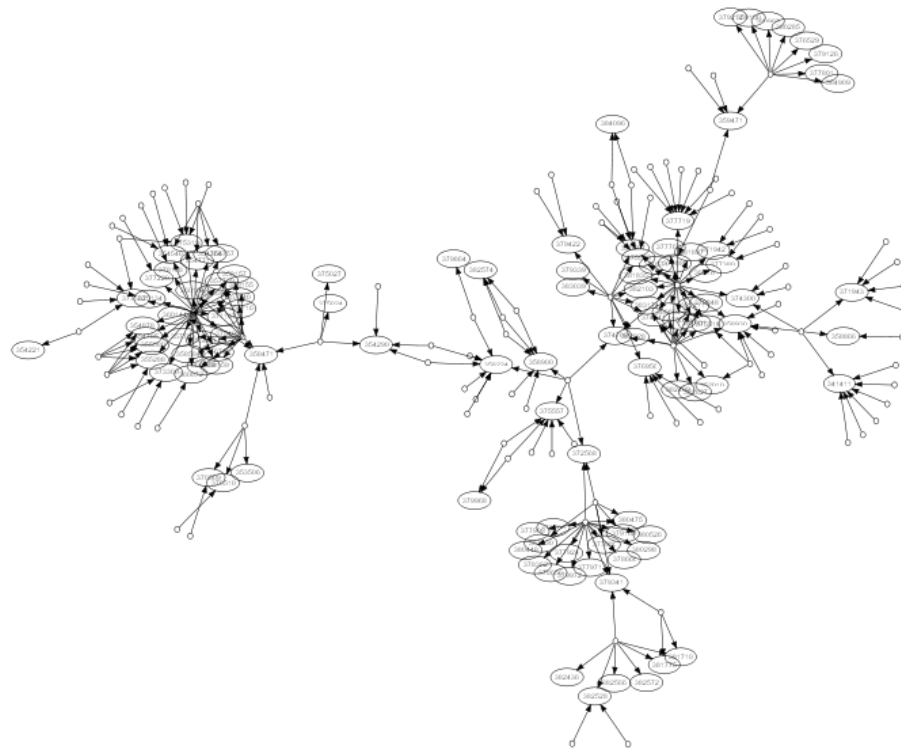


Figure 2.12. GraphViz Graph Visualization (Module Dependencies)

2.4.1. Graph Visualizations

According to the work of Schraefel and Karger [SK06] many good graph visualization algorithms exist for the display of structured (RDF) data. However, the focus is often on visualizing the structure of the data rather than the information itself. Graph visualizations usually provide exclusive global interfaces which offer a convenient way to grab the overall structure. But the usefulness of working with the visualized data is limited. The explicit presentation of the structure is often more relevant for authors (for example of an ontology) than for the users of the structured data who have to perform tasks like *edit*, *compare* or *find* data. Additionally, as they work fully automatically, graph visualizations often have to agree on the least common denominator.

The strengths of graph visualizations are the visual identification of structural aspects, like clusters, high and low interconnectedness in certain areas and an overview of the overall topological complexity, as is the case in the following two examples: Figure 2.12 shows a graph of module dependencies⁷ and the graph shown in Figure 2.13 visualizes a network⁸. When graph visualizations are required to offer overview and information on a single relationship at the same time, the *detail-and-context-problem* arises. The most promising approach to challenge this problem are the *Fisheye* graph visualizations. However, the focus of this work is mainly on non-graph-visualizations. Therefore, in the following two subsections, only two graph visualizations of particular interest are introduced, because they differ from most of the other approaches due to the fact that they have both, a local and a global interface. Please refer to [HMS00] for a detailed discussion of graph visualizations.

⁷Image taken from the website <http://www.graphviz.org>

⁸Image taken from the website <http://www.caida.org/tools/visualization/walrus/>

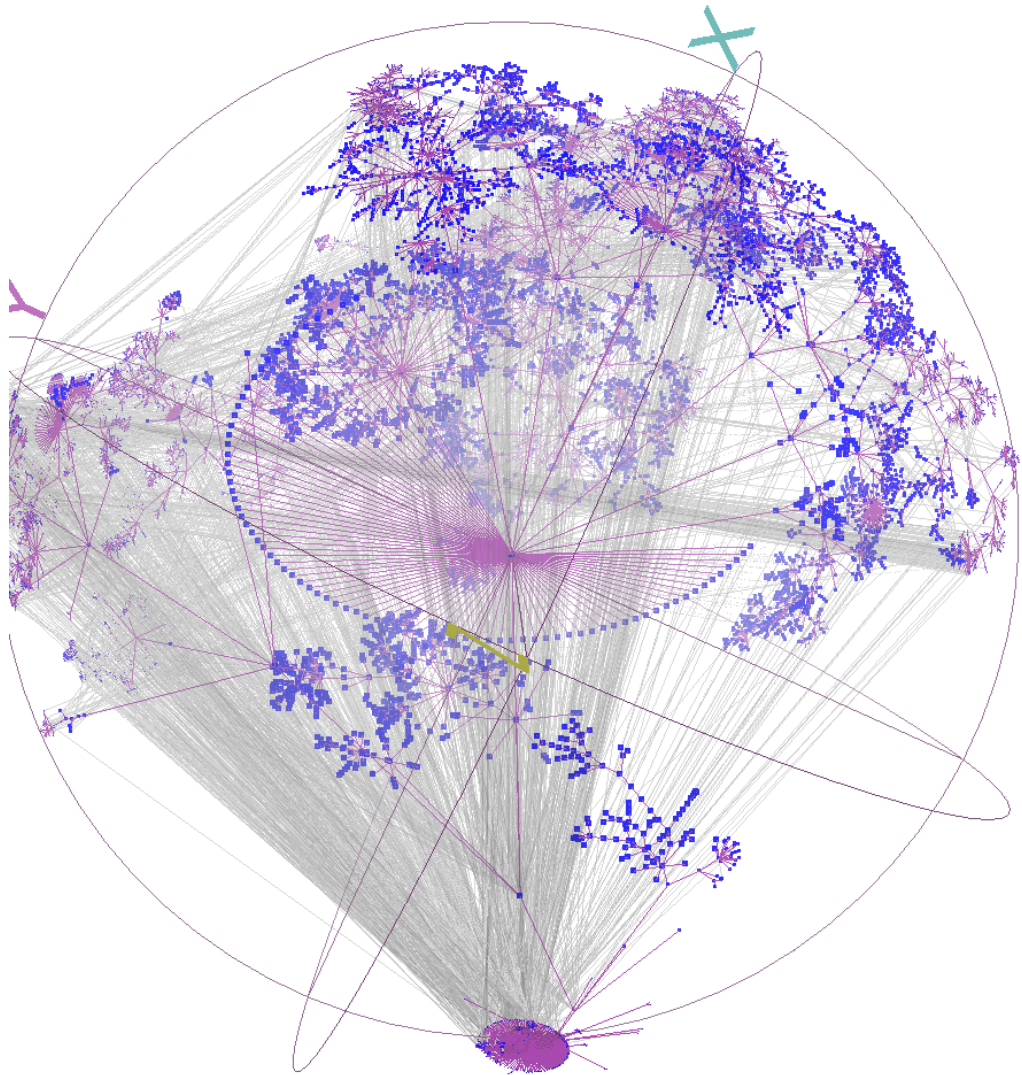


Figure 2.13. Walrus Graph Visualization (Networking)

IsaViz

IsaViz is an RDF graph browsing and editing tool, [IsaViz]. It has its focus on providing a global interface, but instances can also be displayed in detail and it is possible to zoom the displayed graph. The user can modify the appearance of relations and nodes in a declarative way, using either GSS, its own *Graph Style Sheet* language or Fresnel as a more general approach, which will be described in Chapter 4. The styling allows for aggregation of multiple properties and their values to larger visual units instead of drawing a single connection line to each of the values. The combination of the explicit visualization of structure with the presentation of the values themselves leads to a less crowded display, while preserving the complete information content. Figure 2.14 shows a screenshot of the tool.

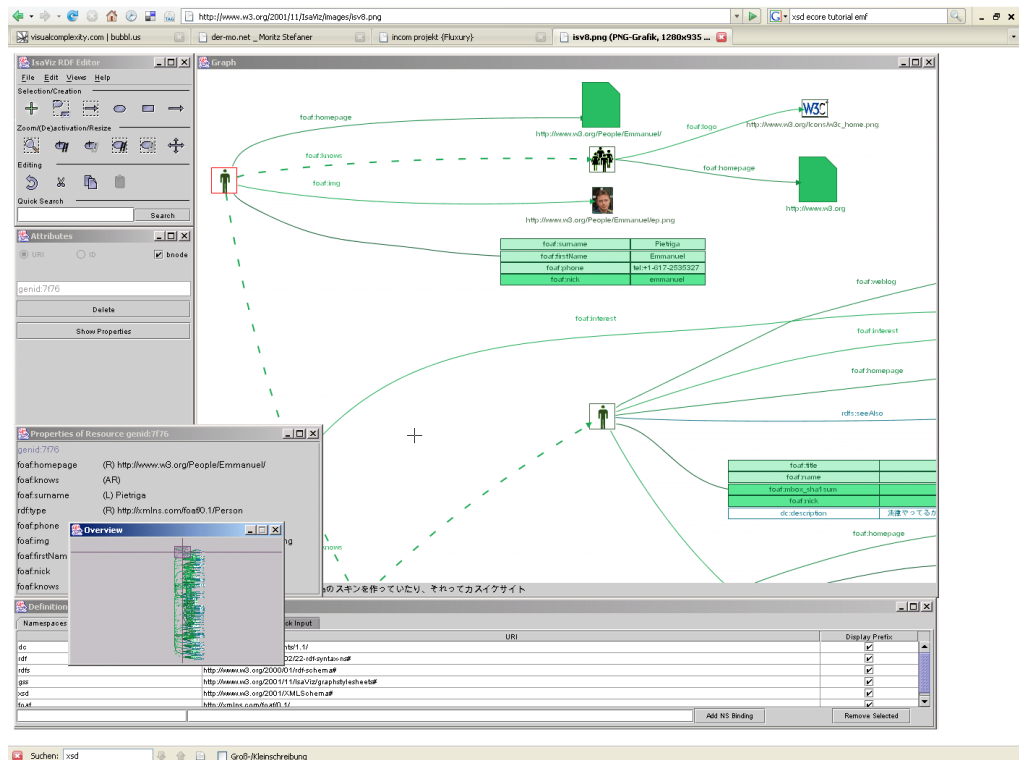


Figure 2.14. IsaViz — FOAF⁹ data

Relation Browser

A small but interesting visualization of RDF data is the *Relation Browser* by Moritz Stefaner, [Relation Browser]. A screenshot is presented as Figure 2.15. It is not a Facet Browser because its intention is not the refinement of a set of results by restricting values, but the exploration of the relations between things. Accordingly, navigation is based on the relationships between the instances. Clicking on an associated object results in centering the selected object and radially rearranging associated objects (click-and-center). It shows always only parts of the RDF graph as a radial tree with a depth of one. Relations to other instances are displayed regardless of their type. The browser focuses completely on instance relationships and shows no classes. Additionally to the global interface, a local interface is supported by the ability of clicking one of the circles. This leads to a continuous magnification of the circle, which then serves as a frame for detailed information on the selected item, this way integrating the global and local interface.

The *Relation Browser* can only be used for an arbitrary domain, after being configured for this domain. This configuration also allows definition of domain specific simple icons and colors to distinguish types of objects and relations. The browser is implemented in *Flash*.

⁹Friend Of A Friend: RDF Vocabulary for the description of people and their relationships

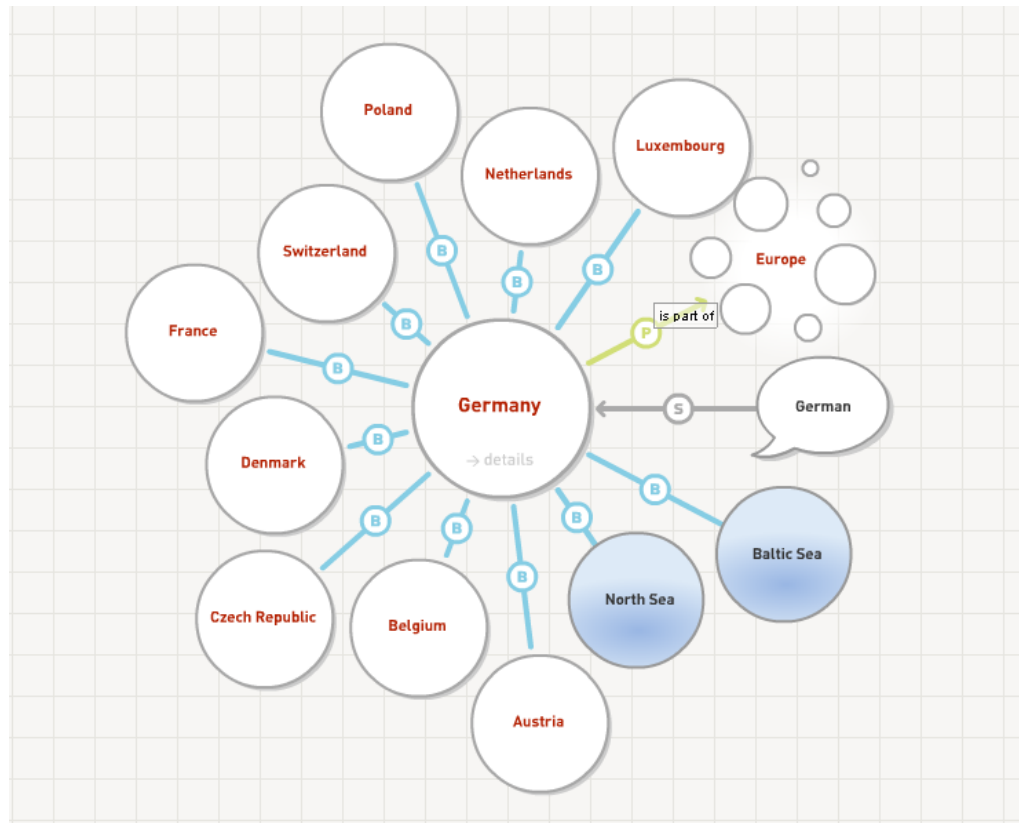


Figure 2.15. Relation Browser — Geographical Data

2.4.2. Faceted Browsers

By the atomic segmentation of a domain, there is the possibility to offer views exactly personalized to the user. Faceted Browsers such as *Longwell*, *mSpace* or *FLAMENCO* are the first affords to use these advantages. They use the Faceted Browsing UI paradigm (c.f. Section 2.3.2) that allows the user to select the results for presentation according to their facets. Finally, the section called “Multiple Categorization in Webshops” investigates the employment of faceted browsing in the field of web shopping applications.

All browsers have in common that they offer a global interface and mostly also integrate a local interface (if not expressly stated different). An initial set of results can be refined by restricting these facets to certain values. All browsers additionally integrate a string search as one of the restrictions.

However, mostly no special widgets are displayed for different facets, nor is the facet structure made explicit. None of the browsers can restrict arbitrary facets to a value range, e.g. the restriction of the facet *age* from *10* to *12* years. This would be a useful for the flexible selection of simple data types. Mostly, even multiple selection of facet values at the same time is not possible (e.g. *10*, *11*, *12* years). With one exception, the facet values are not explained to the user via the user interface and mostly it is not possible to customize the UI by adding facets to the set of restrictable facets or removing them from there.

Often the sorting of results by arbitrary facets is not possible, but only for fixed properties like *Price* or *Relevance*. If it is possible, the facet set for filtering can always be different from the facet set for sorting.

Furthermore, all browsers have in common that they use a two dimensional display paradigm, mostly lists. The implementation is done as a webapplication rendering XHTML and *JavaScript* in all cases where not stated explicitly in the description of the browser. Most of the browsers are not domain-independent, but have to be configured for a particular domain.

The following sections describe some Faceted Browsers, focusing on their special features.

FLAMENCO

FLAMENCO (FLexible information Access using METadata in Novel COmbinations) was developed at the *University of California, Berkeley* [FLAMENCO]. It uses meta data to guide the user through a field of information.

The starting screen shows a list of facets with their values to restrict the result set. As soon as the first selection has been done, all results are displayed in tabular style, as shown in Figure 2.16. The left section allows further restrictions and the upper part allows the removal of constraints. The text input field in the top left corner can be used to search by keywords. The strings are equally handled as facet constraints and result in an entry in the list of applied restrictions at the top of the screen. The facet values, which are limited to those, leading to a non empty result set, are assigned the number of results after a further restriction in brackets. FLAMENCO shows the structure of hierarchical facets like *isA* or *partOf* by displaying at first only the most general values and showing children after selecting the parent. A list of children (which are called subcategories irrespectively of the facet) is displayed as a preview, when hovering the mouse over the facet value before actually choosing it. The browser allows sorting and grouping the results by a configurable set of facets. Starting from a given resource that is displayed in detail, it is possible to search for similar items on the basis of its facets values.

The screenshot shows the FLAMENCO interface for 'Nobel Prize Winners' (1901 to 2004). The search results are filtered by 'COUNTRY: Germany'. The left sidebar contains several facets:

- GENDER (group results):** male (44)
- COUNTRY: all > Germany**
- AFFILIATION (group results):** Berlin University (1), Locarno Pact (1), Germany (38)
- PRIZE (group results):** chemistry (17), literature (6), medicine (8), peace (3), physics (11)
- YEAR (group results):** 1900s (12), 1910s (10), 1920s (11), 1930s (10), 1940s (1)

The main content area displays 'Items 1 to 40 of 44 results' grouped by country. Below the facets, there are two rows of portrait images of Nobel Prize winners, each with their name and dates:

- Adolf Butenandt (1903-1995)
- Adolf von Baeyer (1835-1917)
- Adolf Windaus (1876-1959)
- Albert Einstein (1879-1955)
- Albrecht Kossel (1853-1927)
- Carl Bosch (1874-1940)
- Carl von Ossietzky (1889-1938)
- Eduard Buchner (1860-1917)

Figure 2.16. FLAMENCO - Data on Nobel Prize Winners

Longwell

Longwell is a Faceted Browser inspired by FLAMENCO and developed as part of [SIMILE], which is a joint project conducted by the MIT Libraries and MIT CSAIL that bundles many semantic web projects.

Longwell is intended to present and browse RDF data. As a definition language for the presentation knowledge, [FRESNEL] is used and extended which is dealt with in detail in Chapter 4.

The start screen of *Longwell* (c.f. Figure 2.17) shows a list of classes which have been defined as Primaries. The selection of one of these classes restricts the type of the results to the accordant Primary. This start screen is a kind of a selection by a standard classification before leaving the user to free faceted browsing.



Figure 2.17. Longwell — Primary Selection View

On the browsing view of the *Longwell* user interface, shown in Figure 2.18, the right column is used to represent the facets with their values, which can be used to form restrictions. A click on a certain value restricts the facet to exactly this value. However, it is possible to add more values with the help of a link right behind the summary in the upper part of the screen. Here it is also possible to completely remove a restriction.

As in FLAMENCO, it is possible to sort and group results by sets of facets, which can be configured independently of the facets that are displayed for filtering.

Longwell is a fully domain independent browser for arbitrary RDF data, since it can be used without being configured for a particular domain. However, it can be customized for special display purposes and domains on several levels and offers mashing functions to connect the data to the SIMILE *Timeline* (c.f. Table 2.3) and *GoogleMaps*¹⁰.

¹⁰A geographic mapping service provided by Google: <http://maps.google.de>

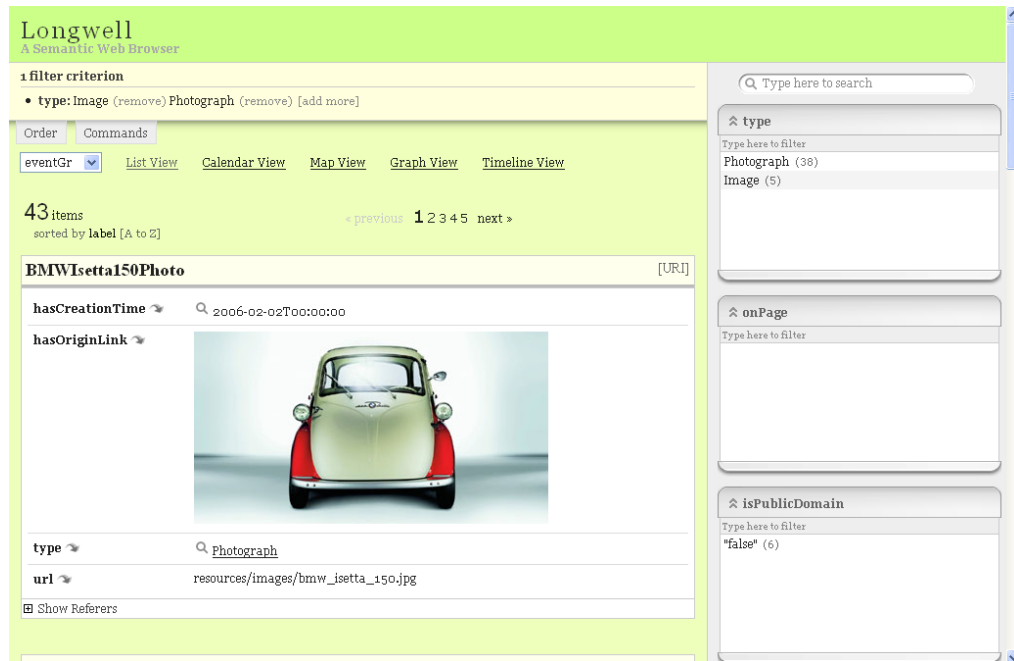


Figure 2.18. Longwell — Browsing View

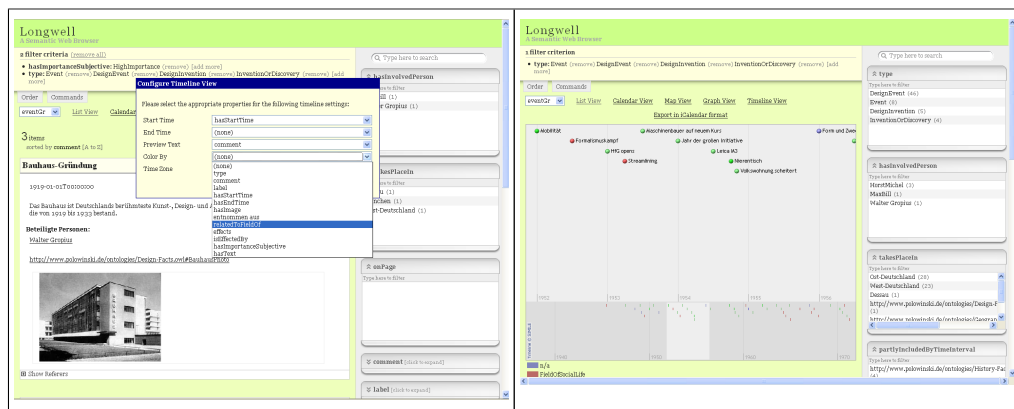


Table 2.3. Longwell — SIMILE-Timeline View

mSpace

mSpace was developed at the *School of Electronics and Computer Science* at the *University of Southampton*. It differs from both *Facet Browsers* described before, in so far, that it does not constantly present the search results, but rather the facet values themselves are items of interest. So it is the only *Facet Browser* that explains the facet values.

Figure 2.19 shows the user interface of *mSpace*. The content window on the lower part of the screen constantly offers information on the currently selected facet value. A special role in *mSpace* plays the *order* of the facets shown on the upper part of the screen. A facet box on the left side always restricts the available facet values of the boxes on the right. This way, the order of the facets reflects the chronological order in which the restrictions were applied. Although this forced order leads to a hierarchy that users are used to from documents providing some benefit, *mSpace* could work in a more flexible way without this order. E.g. a drawback of this approach is the fact that clearing a restriction abandons all other restrictions that were subsequently made. A free clearing and setting of restrictions is not possible. The restriction can be broadened by selecting multiple facet values.

The user can customize the interface by adding new facet boxes from the repository at the top of the screen or removing them again.

Facet values are limited to discrete values (it is not possible to select a value range) and generally displayed as flat facets without revealing the facets structure. An exception is the display of one facet by multiple columns as it can be seen in Figure 2.20. Here the facet *Genre* is split into *Super Genre* and *Genre* to display the structure of this facet explicitly at least to a depth of two.

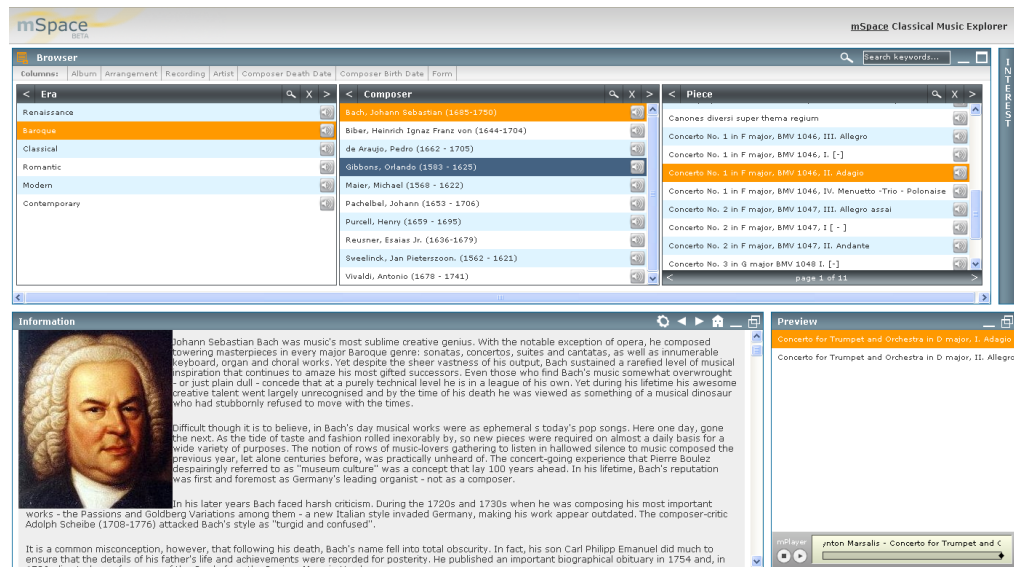


Figure 2.19. mSpace

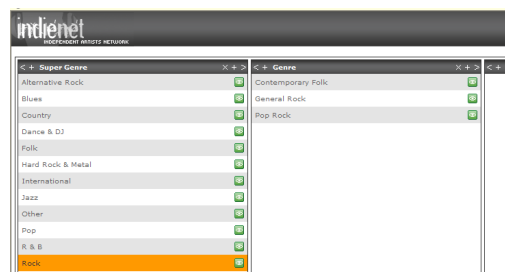


Figure 2.20. mSpace (Multiple Columns per Facet)

FacetMap

FacetMap [SCM+06] aims at providing a fully domain-agnostic search interface for personal documents.

The approach was developed as an alternative to the text list paradigm of most search interfaces and since that, aims at a fully graphical interface that integrates the representation of facets and facet values with the representation of the search results. Whenever possible, graphical icons of the data are used as previews instead of text. But although *FacetMap* includes the results in the display, it offers exclusively a global interface, since it does not provide a detailed view on single instances.

Figure 2.21 shows a screenshot of the application. The screen is split into two parts after the first narrowing of the data set. The left part shows the still unrestricted facets, the other displays facets that have already been restricted to some value. The facets on the left show only the values that are still applicable.

A variant of the treemap algorithm is used to fill the available screen space with rounded rectangles and ellipses to display facets, facet values and results. To calculate the appropriate amount of screen space that a facet may allocate, facets are ranked based on their metrics and their relatedness to the already restricted facets. The problem of guaranteeing a minimum size of display items, while still showing all possible facet values at the same time, is tackled by the use of an *Overflow Bucket* that serves as a place for the rest of values that can not be displayed by at least one ellipse.

The way that *FacetMap* nests ellipses and rectangles is able to represent the structure of a single facet explicitly.

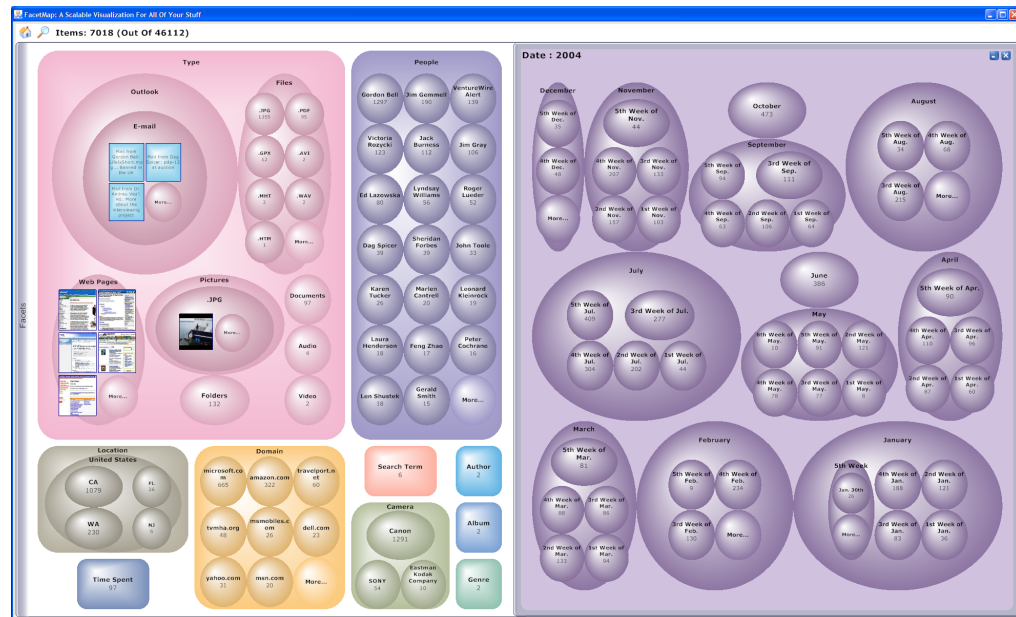


Figure 2.21. FacetMap

Multiple Categorization in Webshops

Recently, companies such as *Ebay*¹¹, *Amazon*¹² and especially *Google Base*¹³ exercise the option of multiple categorization to simply categorize their products and allow for faceted search in their product data.

All three use a combination of a normal classification system with a faceted system in order to reduce the amount of very general queries and to inspire the user by offering a starting point for search. Besides that, the string search by keywords is possible as with all Faceted Browsers. After roughly choosing a product in a standard hierarchy, or performing an initial string search, the client can concretize his search by selecting several facet values. Another commonality of all faceted browsing solutions in webshopping is the ability to search for similar items starting from the detailed view of a product.

Ebay offers a fix set of all facets for a category with all possible values. After choosing a certain value for a facet, the set of facets is *not* reduced to the ones that still offer results, because all restrictions are evaluated altogether only after the submit button is pressed. Although this offers a more consistent overview of all restriction possibilities, it has the drawback of possibly returning a zero result set. Already in this standard search, *Ebay* offers possibilities for the user to basically customize the browsing interface by

¹¹<http://www.ebay.com>

¹²<http://www.amazon.com>

¹³<http://base.google.com/>

removing or adding facets. However it is not possible to define a value range, except for fixed ranges of prices. Three different widgets are used: Text input fields, checkboxes and option lists.

A fully faceted alternative user interface for Ebay has been developed under the name *Ebay Express*¹⁴. The user can choose from about 20 main categories or choose a category containing products matched by an initial string search. Within the subcategories that are displayed, multiple facets with some values and value ranges are proposed for a restriction of the results. The facets are ranked by relevance and the values are assigned a cardinality of the result after a potential restriction. More facets, called *options* can be added and also multiple selection of facet values is allowed. Again, only some important facets, as the price, have special widgets to allow for manual input of a range for example. Also the sorting of the results is still limited to a fixed set of facets like price or relevance, as in the standard Ebay solution.

The results can be displayed as a list or as a table (c.f Figure 2.22).

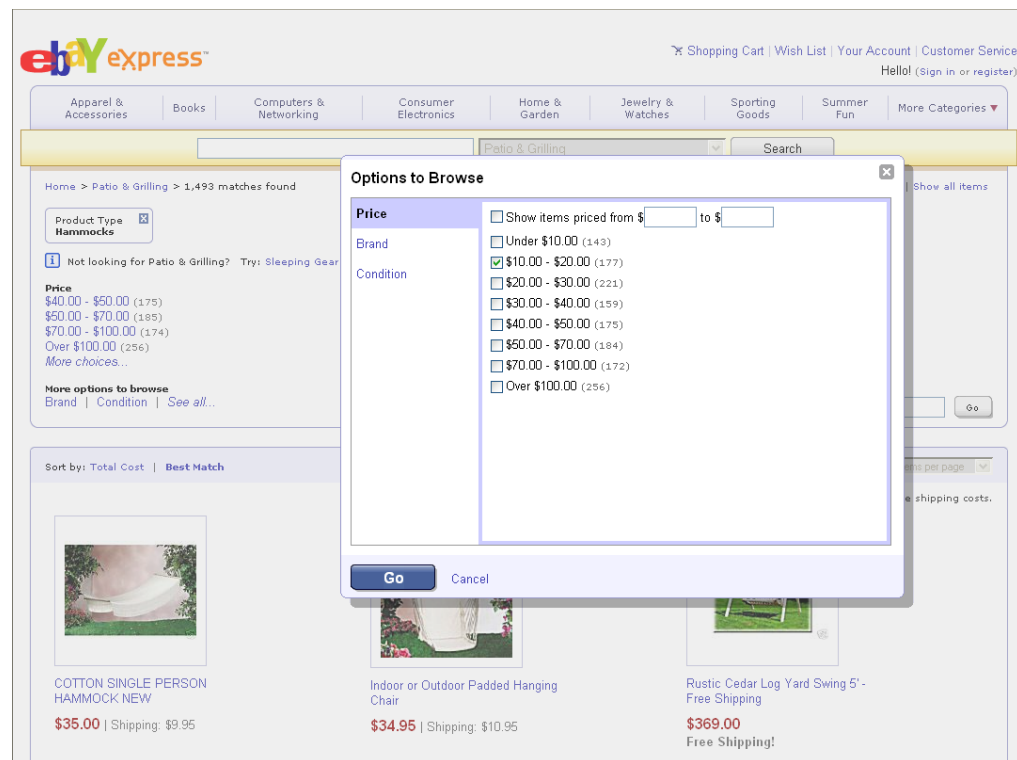


Figure 2.22. Ebay Express — Search for a Hammock

The interface of *Amazon* is basically oriented at a standard classification system. Figure 2.23 shows the selection of books for children at the age of at least 10 years. A mixed hierarchy is used to display the choices of the user, containing values from multiple facets and the facet names themselves: After choosing the category *Books for children* (belonging to the facet *Genre*) and subsequently deciding to further restrict the results by the facet *Age*, the user has chosen the facet value *starting from 10*. All this is joined to one tree which also serves to make the structure of the facets explicit. Result cardinalities are displayed in brackets after each facet value that is not used for restriction so far. However, some facets can be restricted independently from the others; Again the facet price is handled exceptionally, but also facets like *Brand* are treated separately, depending on the field of products.

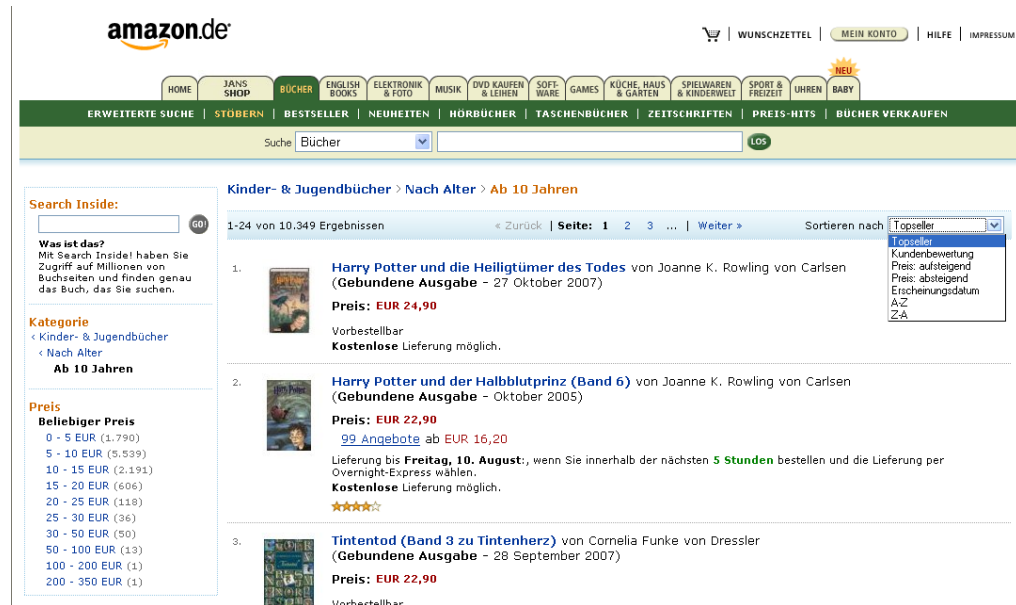


Figure 2.23. Amazon — Books for Children over 10

A first analysis of *Google Base* [Google Base] as a faceted system has been done by [Sin05], though the results are only informally published and some things have been changed at *Google Base* in the meantime. This work served as a base for our review of *Google Base*.

Google Base collects data, including product data, and makes it searchable by faceted browsing . It allows for complex filtering of the results of an initial string search or the selection of a main category. No subcategories are displayed, yet there are more than the twelve *Popular Classes* and additional *Interesting Items Types* that are initially shown, because the author can flexibly define his own categories.

As soon as a category is displayed, the system proposes suitable facets for restriction and for sorting. Already made restrictions can be cleared by setting the value back to *any*. *Google Base* allows for sorting the results by several, yet selected facets. Complex specialized widgets are used to allow the combined input of fixed, predefined and user-defined value ranges or enable multiple selection, for example.

Depending on the category, a domain specific view is offered, e.g. the category *houses* starts with a view that integrates *Google Maps*. The faceted system allows for easy build of mashup applications like this (c.f. Figure 2.24).

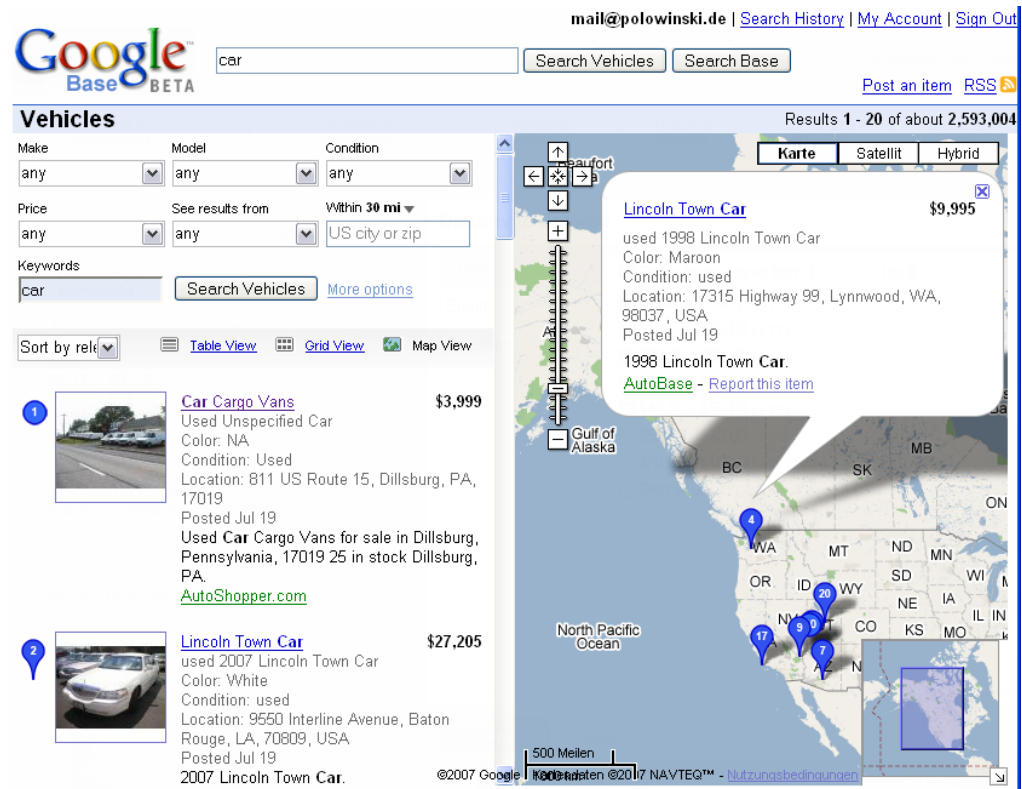


Figure 2.24. Google Base — Search for Cars

Mambo

Mambo (Mobile Aspect-based Media BrOwser) is a browser developed at the *Dresden University of Technology*, [Mambo].

The interface is designed to browse large media collections, with special focus on the usability on mobile devices. Since that, the integration of context and detail is stressed and implemented by a novel zooming widget called *AspectZoom* (a generalization of the *TimeZoom* widget [DW06]). So far the interface completely renounces the use of string input and relies on the zooming metaphor to establish "a consistent way of searching, browsing and filtering data".

Although it has many commonalities with Faceted Browsers, it is called an *Aspect Browser* and actually is not a Facet Browser in the sense of the Faceted Browsing paradigm. Facets play an important role in Mambo, but the interface is centered around *one* selected facet that is restrictable at any time. The restrictions to the facets that *Mambo* offers for browsing (c.f. Figure 2.25, bottom right corner) can not be combined to form a single result set as in the other browsers.

However, *Mambo* could be modified to combine multiple widgets for many facets simultaneously. Another difference is the fact that it is the only browser studied that explicitly visualizes the structure within one facet (except for *FacetMap* and classical tree representations in the standard classification part of some browsers). Its *AspectZoom* widget is specialized to the visualization of tree structures and lends itself to be integrated with Faceted Browsers, too.

Mambo interprets facets with a string type as structured by a tree instead of a list. This eliminates the need to display long selection lists and replaces it with a less space consuming alternative. Although one could argue, that string completion could be a similar and, assumed the availability of a keyboard, the better choice in some cases, there

is a variety of facets that could be efficiently displayed with the *AspectZoom* widget, including *partOf* relationships (e.g. for time spans, geographic regions or body parts, see also the section called “Reflecting the Facets Characteristics”).

The ease of use is additionally improved by a click-and-center behavior, that reduces the necessary clicks to a minimum.

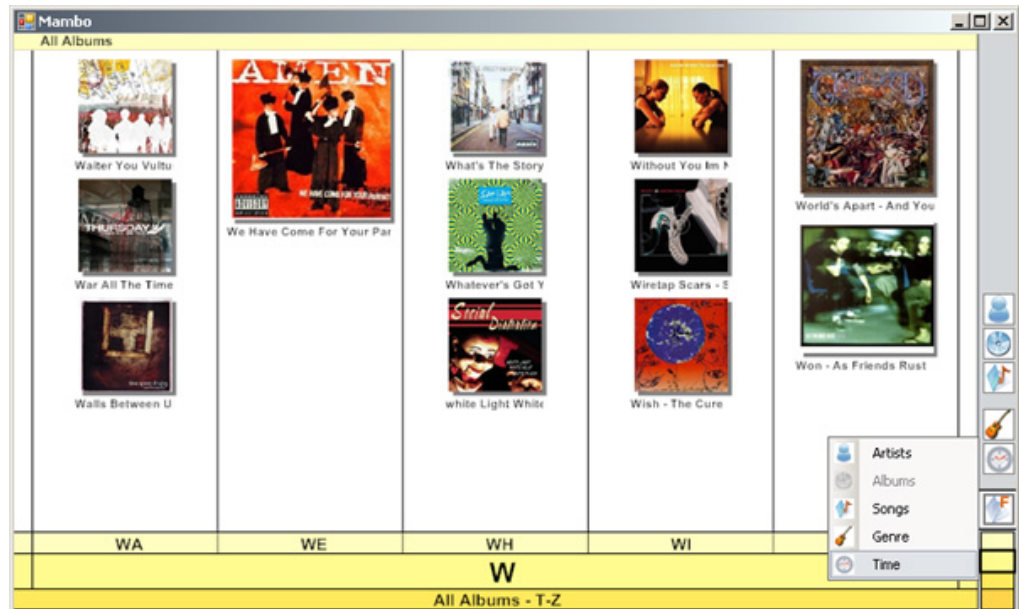


Figure 2.25. Mambo — Aspect Browsing of Music Data By Title

2.5. Categorizing SemVis

In this section we classify SemVis with the help of the before introduced categorization criteria for visualization systems and compare it to the existing solutions mentioned above.

SemVis covers the whole visualization process. Its main focus is on the support of the formatting step, but the support of the preceding selection step (the filtering) and the necessary structuring are also dealt with and visually supported by the system.

Another criteria of visualization systems is the degree of automatism and, related to that, the degree of domain independence. Whereas SemVis aims at fully domain independence but a semi-automatic mapping and structuring procedure, some systems try to offer a fully automatic presentation of arbitrary data, e.g. the Faceted Browsers *Longwell* or *Noadster* [Rut05]. The domain independence without configuration is not the goal of SemVis.

The selection step is done visually in SemVis by Faceted Browsing. It is well-suitable to implement the filtering ideas of the old framework. The following characteristics of Faceted Browsers are fulfilled by the system: SemVis offers the possibility to clear and add restrictions to facets, including the selection of multiple facet values and the ability to choose value ranges. The use of specialized widgets for different facets which explicitly show the facet structure is supported as well as the ranking of facets with the help of metrics, the use of presetting and the customization of the user interface.

The focus of SemVis is on providing a global interface. The visualization of single characteristics of the data allows for an overview of the data on a global level. However, this does not necessarily require the explicit display of relations as in graph visualiza-

tions. This can be optionally chosen. Rather the consequent mapping of facets to Visual Variables to transport information supports the global interface. This is something that none of the studied facet browsers does, with some exceptions (for single aspects) such as the SIMILE timeline that is mashed up with Longwell or the integration of Google Maps in Google Base. This classical cartography of data, using Visual Variables is done by graph visualizations however (mostly using exclusively spatial dimensions and color). SemVis combines the Faceted Browsing paradigm and the classical visualization into one.

Depending on the output format SemVis can also generate an integrated interface. For example, SemVis's 3D graphic output allows for an integrated view of structure and detail by presenting a coarse grain, low detailed overview, which can be zoomed in, making use of a *Level of Detail* (LOD)¹⁵ mechanism. Details on a single instance, including a description and a textual representation of selected properties values, can be found at the highest LOD.

Through a flexible mapping mechanism, SemVis is not limited to a specific display paradigm, but the generated structure can have many shapes, and includes, aside from tables and lists, also 3D variants.

A complete classification of many visualization systems for RDF data, compared to familiar text document principles, can be found at [Rut05].

After having categorized SemVis in the existing solutions for the visualization of structured data and having introduced general terms and techniques, the next chapter elaborates the exact requirements for a semi-automatic system that can flexibly generate customized visualizations for RDF data.

¹⁵Level of Detail — The principle of increasing detailed variants of a displayed object, depending on a criteria such as the distance of the viewer to the object

Chapter 3. Requirements for a Flexible Visualization

This chapter describes the staged process for visualization that SemVis proposes, including the configuration of mapping and filtering by the GUI. Since this process involves many *Actors* and use cases we shortly give an overview of them, without specifying the details of the mentioned knowledge repositories and definition files. This will be done in Chapter 7.

3.1. Actors

SemVis assumes the following three actors to be involved in the visualization process:

The *Programmer* uses and extends the framework to realize new visualization structures for example. If she adds a new visualization platform she also has to describe this platform to the system.

The *Admin* is, for instance, a technician in a museum. Her task is to define a presetting of mapping and filter settings, suited to a special domain, that was not known at the time of programming. She adds the presentation knowledge and configures SemVis by writing definition files, or preferably through the *Configuration GUI*. The GUI enables the Admin to configure the visualization structures. She is able to save and load configurations. In this way, it is possible to develop many configurations for different purposes and compare them easily.

The *User* is, for example, a visitor of the museum or a spectator of a generated website¹, who views and explores the data. It should be possible for him to specialize his view by altering the mapping and refining the filters. Doing this, she is supported by the systems *User GUI*, which can be realized in the output format, or by building a frame around the presentation. The User should also be able to save her settings and later restore the old state of exploration.

3.2. Use Cases

The following all-embracing scenario could take place in order to map a domain to a final concrete visualization. Roughly spoken visualization and data are annotated with meta data, the system is then able to make suggestions and the Admin can choose from these prechosen possibilities. The scenario described here assumes the existence of a configuration GUI, which is crucial for the full value of SemVis, since the meta information on the structures, platforms and the general graphics knowledge can only be offered to the involved actors via a GUI.

We assume the following scenario: The SemVis framework shall be used to visualize semantic web data about a specific domain (available in RDF), *Extensible 3D [X3D]* shall be added as a new visualization platform and a new visualization structure needs to be described.

3.2.1. Extending SemVis (Programmer)

Figure 3.1 shows the steps that have to be taken by the Programmer:

¹In a scenario, where a web user visualizes data for herself, the role of the Admin and the User may also be played by one person in two subsequent steps.

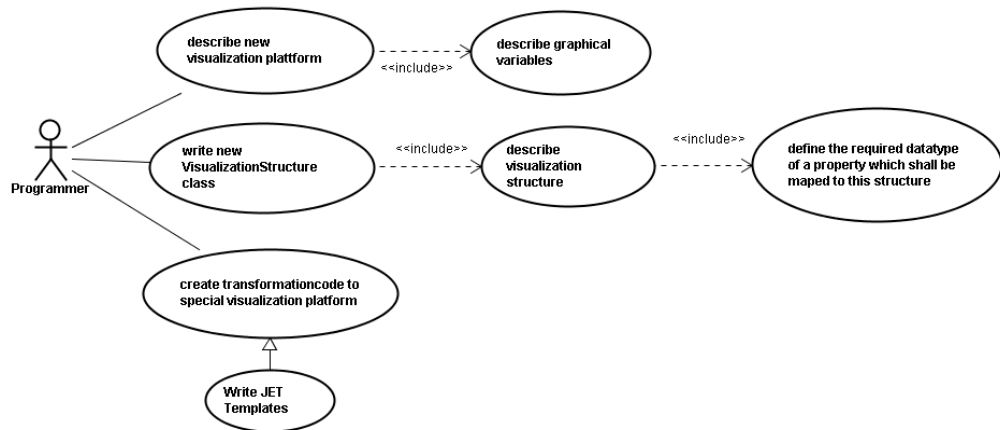


Figure 3.1. Use Cases — Extension by the Programmer

1. The Programmer describes the new visualization platform in the definition file.
2. She extends the framework by describing the new visualization structure.
3. She implements the algorithm building the structure and links the description to the newly written Java class.
4. She creates a set of (nested) templates for the new presentation platform.

3.2.2. Configuring SemVis (Admin)

Figure 3.2 shows the steps that have to be taken by the Admin to configure SemVis. The required order of these actions and the necessary reaction of the system are specified in the corresponding Activity Diagram in Figure 3.3.

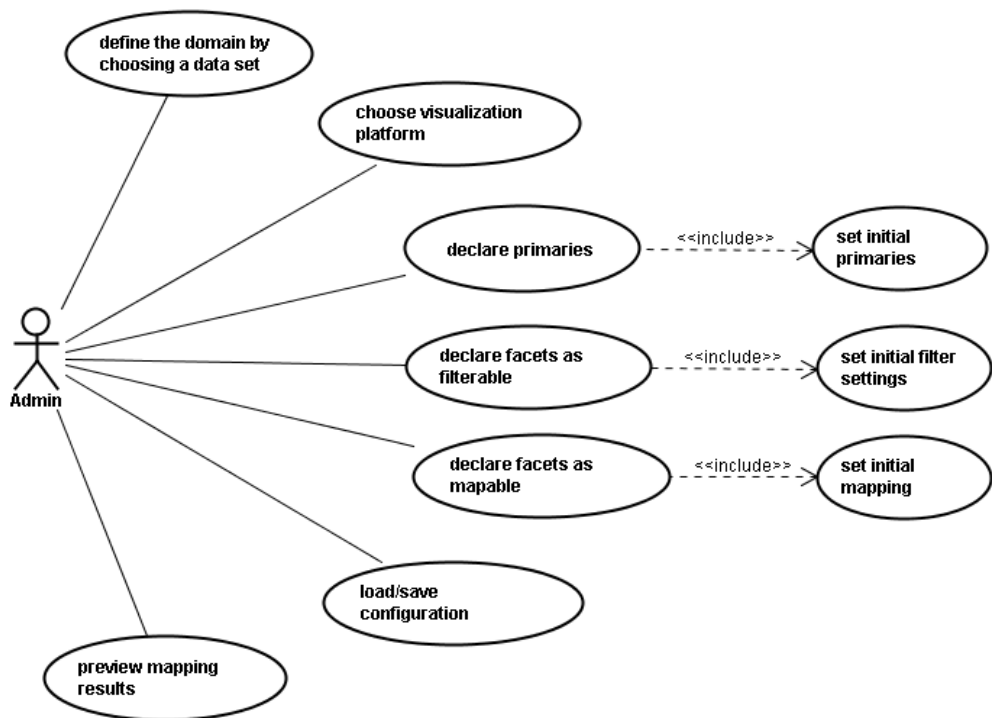


Figure 3.2. Use Cases — Configuration by the Admin

The first step has to be taken by the system. It offers the Admin the possibility to load existing configurations as a basis for further refinement (a). After that, the Admin can define or change the domain and choose a visualization platform (b,p).

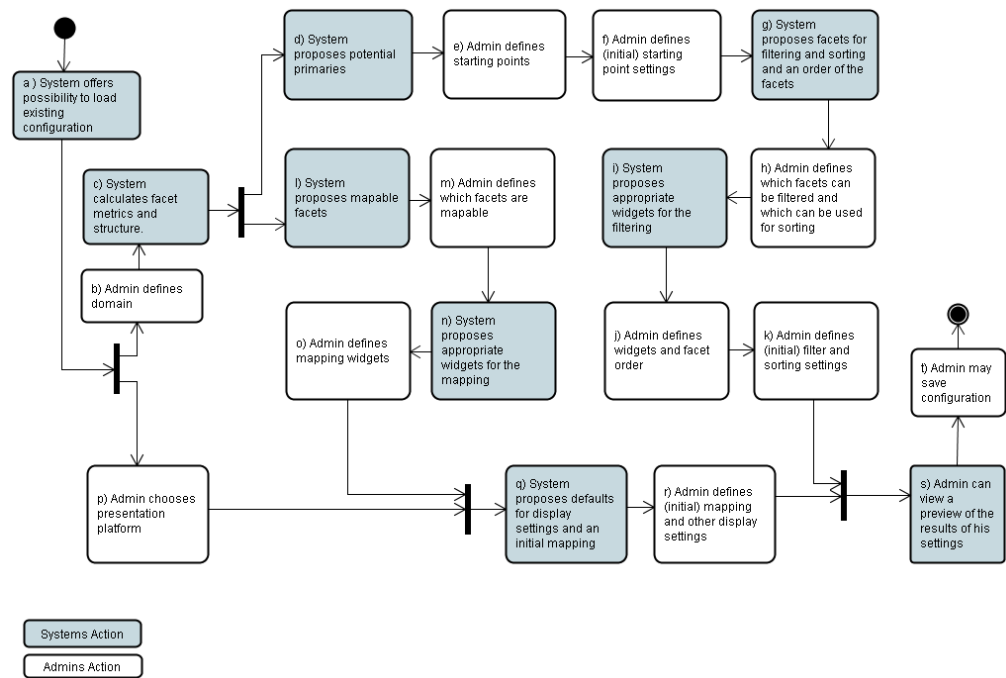


Figure 3.3. Activity Diagram — Configuration by the Admin

As soon as the domain is known, the system can start calculating metrics on the facets and find out about the structure of the facets (c). This knowledge is necessary for later suggestions to the user, such as the proposal of potential starting points (d). The suggestions of the system, regarding the starting points, can be ignored or refined by the Admin in the following step (e). After having defined the possible starting points for the User, an initial setting for starting points is defined (f). Now, that the classes of primary interest are defined, the system can examine the properties of these classes and suggest facets that are most suitable for filtering and sorting, as well as an order for the facets, based on the knowledge from step c). The order of the facets can be chosen according to their quality as a *Navigator* (c.f. Section 2.3.2). The facets for filtering and sorting may differ (g). Again, the Admin can reduce or extend the set of suggested facets for filtering and sorting (h). In step (i), the system proposes suitable widgets for each facet which can be replaced or confirmed by the Admin in step (j). These widgets can already be used to conveniently define the initial filter settings for the User in step (k).

The mapping configuration is done in a similar way. The process starts with the proposal of well-mappable facets by the system (l), again based on the knowledge from step c). These suggestions are refined by the Admin (m). Now the system can propose appropriate mapping widgets for each facet (n). These can also be exchanged by the Admin (o). Before the proposal of display settings (i.e. the initial mapping) by the system (q), it must be previously defined, which visualization platform is targeted (p). The system needs this information to prefer those mappings, which are best suited to the chosen platform. Besides the initial mapping, defaults for other display settings (styles of textual information such as *font-family* and *font-size*) are presented to the Admin for optional editing (r).

When both filter and display settings are completed, the Admin can choose to preview the results of the configuration (m) and save it (n).

3.2.3. Browsing with SemVis (User)

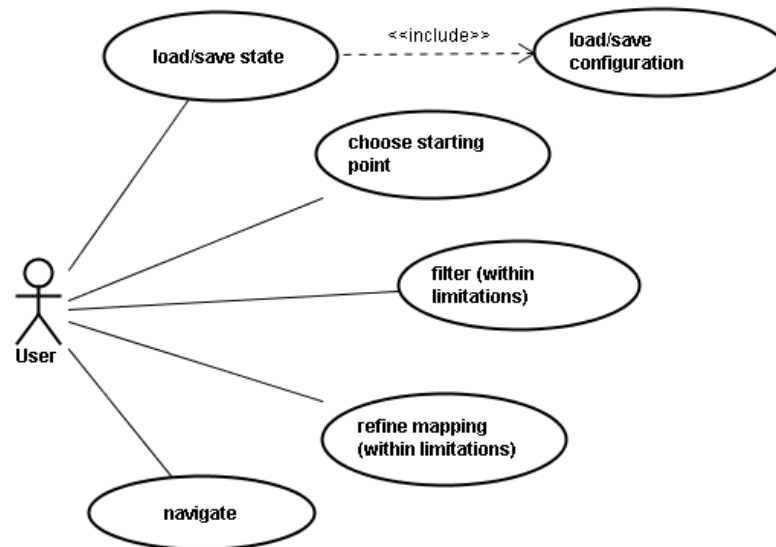


Figure 3.4. Use Cases — Browsing of the User

The X3D output generated by SemVis might be shown in a museum, eventually in a CAVE² in 3D or simply on a personal computers screen, loaded from a website and rendered with an X3D browser. As shown in Figure 3.4, using SemVis comprises the following actions:

1. The *System* loads the initial mapping and filter settings defined by the Admin.
2. The User chooses a starting point.
3. The User can modify the mapping (within the limitations defined by the Admin).
4. She can then modify the filters for the defined primaries (within the limitations defined by the Admin)
5. The User can navigate through the scene or document.
6. The User can finally save her state including her personal mapping, view point and other personal settings.

When saving the scene, SemVis has to save the chosen starting points, the facets and filter settings, the mapping configuration and also the current view point, if the view point can be changed.

²CAVE — Cave Automatic Virtual Environment. A cubic room with projectors directed to most of its sides to enable an immersive virtual reality.

Chapter 4. Fresnel, a Standard Display Vocabulary for RDF

The *Fresnel* specification aims at providing a general purpose standard RDF display vocabulary that can be used instead of many proprietary approaches, which have been developed recently. The vocabulary is supported by the W3C and was released in 2005, [FRESNEL]. It is written as an ontology itself. As such, it follows a fully declarative way and can easily be extended. Furthermore, it is developed with the explicit intention not to be limited to boxes (as used on CSS-styled XHTML websites) but can also be used for other paradigms such as a graph. This characteristic makes it a good candidate for a uniform description of display knowledge in SemVis, where several display paradigms have to be supported.

The vocabulary consists of a *Lens* and a *Format* part. The Fresnel *Lens* mechanism transforms the graph of the RDF data into an ordered tree and thus takes the structuring part of the visualization steps (c.f. Section 2.3.2), including a selection and ordering on the property level and a mechanism to limit cyclic relationships to other resources. After this serialization, it is well defined which properties are displayed and how the data is ordered. The *Fresnel Formats* assign properties to presentation formats, which do very general formatting and offer mechanisms to further define the appearance with CSS.

SemVis uses the Fresnel vocabulary for defining and storing initial and generated display information. In addition to the advantage of providing a complete solution for the description of display information, the declarative way of Fresnel also served as inspiration for the SemVis mapping vocabulary, which will be described in the next chapter. Due to the ease of Fresnel's extensibility, it can also be considered to turn the mapping vocabulary into a Fresnel extension.

The declarative style supports easy exchange and sharing of the display definitions. Due to the standardization of the language, this is even possible between different tools.

The code examples following each description are all written in the *Notation3* [N3] syntax, as it counts for all examples throughout the thesis unless otherwise expressly stated.

4.1. Fresnel Lenses

The *Fresnel Lens* mechanism transforms the RDF graph into an ordered tree and thus picks a certain view on the data. It defines which properties of the instances, to which the lens is applied, are to be shown and additionally defines an order of display for these properties. The cyclic relationships that exist between RDF data can be broken into a tree by limiting the recursion depth.

```

1 :eventLens rdf:type fresnel:Lens ;
2   fresnel:purpose fresnel:defaultLens ;
3   fresnel:classLensDomain hist:Event ;
4   fresnel:showProperties
5     (
6       hist:hasStartTime
7       hist:relatedToFieldOf
8       mid:hasImportance
9     [
10      rdf:type fresnel:PropertyDescription ;
11      fresnel:property hist:hasInvolvedPerson ;
12      fresnel:sublens :personLabelLens ;
13      fresnel:depth "3"^^xsd:nonNegativeInteger ;
14    ]
15   ) ;
16   fresnel:group :mainGr ;
17 .

```

Example 4.1. Fresnel Lens Definition

An example for a `fresnel:Lens` definition in Fresnel using *n3-Notation* is shown in Example 4.1. It states, that this Lens should be used by default (c.f. line 2) to display resources of the type `hist:Event` (c.f. line 3). Other values are `fresnel:labelLens` or new purposes defined by the developer.

Further on, the Lens definition states that the properties `hasStartTime`, `relatedToFieldOf` and `hasImportance` should be displayed (c.f. lines 6-8).

The property `hasInvolvedPerson` is defined to be displayed via a sublens, namely the `personLabelLens` (c.f. line 12). Sublenses are used to define that an related instance shall be rendered with a specific lens. Figure 4.1 shows the abstract sublens concept of Fresnel (on the left) and a possible instantiation (on the right). A sublens allows for a different display of instances in different roles. While the person *X* is an instance of the primary class *Person* and displayed in full detail, the other *Person* instances are only displayed as items of secondary interest with a possibly reduced, or completely different appearance.

This approach is well suitable for the idea of displaying related objects visually associated to a primary class, rather than connected by explicit visual elements as arrows or connection lines.

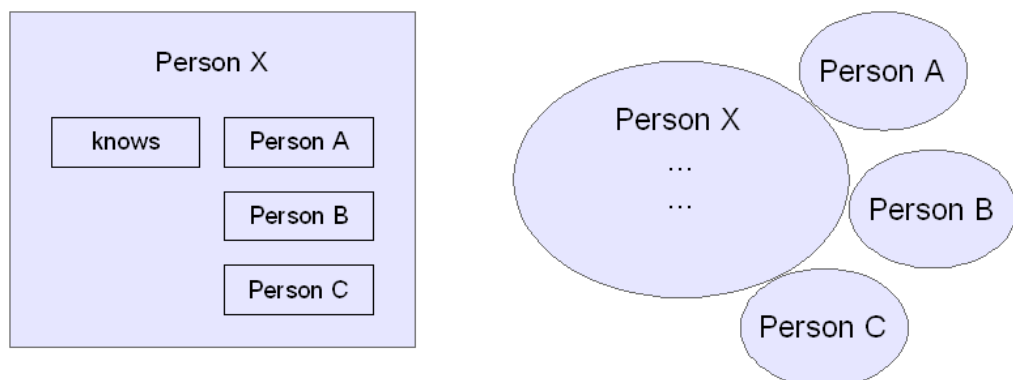


Figure 4.1. Use of Sublenses in Fresnel

The recursion depth for showing related instances with this lens is limited to a value of "3" with the property `fresnel:depth` (c.f. line 13). This is actually not necessary in this case and only defined for demonstration purposes, because the person-

LabelLens does not make references to other Persons, so that it can not be called recursively.

Finally, in line 16, it is stated that the Lens belongs to the mainGr group. The function of Groups is defined in Section 4.3.

4.2. Fresnel Formats

Formats can be used to assign properties to special presentation formats. With the help of a `fresnel:Format` it is possible to define presentation-paradigm-unspecific styling information such as additional content between each value, or labeling of properties as well as the value format (e.g. display as image or as URL). The actual appearance can be determined by CSS-classes which can be bound to a format. The properties can be described in relation to their context, defining different formats for the same property in different situations.

Example 4.2 shows the usage of a Fresnel format to define the format of `hasStartTime` property values. It states, that each `hasStartTime` property should be displayed without a label and its values should be styled according to the style class `date`. This `fresnel:Format` also belongs to the `mainGr` group.

```
1 :hasStartTimeFormat rdf:type fresnel:Format ;
2   fresnel:propertyFormatDomain hist:hasStartTime ;
3   fresnel:label fresnel:none ;
4   fresnel:valueStyle "date"^^fresnel:styleClass ;
5   fresnel:group :mainGr ;
6 .
```

Example 4.2. Fresnel Format Definition

4.3. Fresnel Groups

Groups can be used as a convenience construct to bundle styling definitions that belong to all members of the group including resources, properties, labels and values. It is possible to define common `labelFormats`, `propertyStyles` or `resourceStyles`.

The following listing (c.f. Example 4.3) defines a Group where all properties have a common stylesheet (c.f. line 2) a colon following the label (c.f. line 3) and all properties have a `styleClass` attribute with the value `standardPropertyStyle` (c.f. line 4).

```
1 :mainGr rdf:type fresnel:Group ;
2   fresnel:stylesheetLink "styles/person.css" ;
3   fresnel:labelFormat [ :contentAfter ":" ] ;
4   fresnel:propertyStyle
5     f"standardPropertyStyle"^^fresnel:styleClass ;
6 .
```

Example 4.3. Fresnel Group Definition

4.4. Primaries (Starting Points)

The definition of Primaries, the starting points and basis for further refinement by faceted search as described in Section 2.3.2, is already part of the Fresnel vocabulary and can be done within the Fresnel definition of a group (c.f. Example 4.4). In Fresnel, a class that is defined by the property `fresnel:primaryClasses` can be directly

displayed on the root level. Other classes may only be used with sublenses. Example 4.4 defines that only the class `hist:Event` is handled as a *Primary*.

```
:mainGr rdf:type fresnel:Group ;
  fresnel:primaryClasses (
    hist:Event
  );
.
```

Example 4.4. Definition of Primaries

4.5. Selectors and Inference

Selectors are needed to specify sets of instances or properties. Three variants of selectors exist in Fresnel. Besides the *simple selectors*, that can select classes, a selector language can be used to specify a set of instances. Either *Fresnel's* own language, the *Fresnel Selector Language* [FSL], or SPARQL as a standard selection language, can be used for selection.

Please note that the selection performed with the selectors is not the selection described before as the first step of a visualization process (the filtering of resources). Selectors are responsible for the selection of potential instances, to which a lens or format applies and for a selection on the property level to define the properties that are to be shown for a resource, handled by a lens.

The Fresnel vocabulary has no support of defining inference, but delegates this to the inference layer of the RDF repository. For example, there is no support to define on the Fresnel level if subclasses should be considered or not. However, some inference can be stated with the help of the FSL, but this is limited to the *subPropertyOf* and *subClassesOf* relationships. It can be stated, that a certain property, including its subproperties is chosen, by prefixing the query with the character "^". Section 4.5 [34] gives an example¹ for this.

```
# properties to be shown for resources handled by this lens:
# all foaf:knows properties
# as well as properties declared as subproperties of
# foaf:knows (in the associated schema)
^foaf:knows

# exampleLens' domain: all instances of class foaf:Person
# (or any of its subclasses) older than 60
^foaf:Person[ex:age/text() > 60]

# properties to be shown for resources handled by this lens:
# foaf:knows and any subproperty of foaf:knows,
# provided that the value is an instance of class
# foaf:Person or one of its subclasses
^foaf:knows[^foaf:Person]
```

Example 4.5. Inference in Fresnel

4.6. Application and Reusability

As mentioned above, Fresnel is presentation paradigm agnostic and not limited to the box paradigm. As an example IsaViz employs Fresnel descriptions to define the style

¹Taken from the FSL reference: <http://www.w3.org/2005/04/fresnel-info/fsl/#rdfsowl>

of its graph layout (since Version 3.0). Display definitions written by SemVis in Fresnel can be read by applications such as IsaViz or the Longwell browser of the SIMILE project. Other browsers, that employ Fresnel are the SWT RDF browser and Fresnel editor *Cardovan* (discontinued) and the PHP RDF browser *Horus* (discontinued) [BLP05]. More projects and engines that employ the vocabulary are currently developed, such as the domain specific *Geonames Browser* for geographical data, which uses Fresnel and Fresnel extensions, [Geonames Browser].

In addition to the benefit of being paradigm agnostic, the modularization of Fresnel supports reuse and portability. The specification is not only modularized into a lens and format part, but also into a core and an extended part. The extended vocabulary allows to specify alternate properties and to merge similar property values. Other features cover support for different output-types, direct linking to CSS stylesheets and lens inheritance. This modularization allows applications that implement Fresnel to optionally support the extended part, guaranteeing a minimum of exchangeability between all Fresnel applications.

4.7. Implementation

For the processing of Fresnel definitions there are two Java implementations. The first is used within the Longwell browser and is currently used in SemVis. This implementation is referred to as *Fresnel Engine* below. The other is called [JFresnel] and was recently developed as a standalone project. Additionally, there is a PHP implementation called Arago [GH05].

The *Fresnel Engine* produces a tree of Java objects such as the example result tree shown in the Object Diagram Figure 4.2. A `Result` is assigned one `PropertyResultSet` which contains a list of `PropertyResults` and `NoSuchPropertyResults`² for every property listed in the Fresnel definition. Each `PropertyResult` is assigned one `LabelResult` and one `ValueResultSet`, which contains the instances of `ValueResult`.

The engine performs a recursion, which can be limited in depth, to create the values, that are results themselves. In this case `ValueResult` wraps another `Result`.

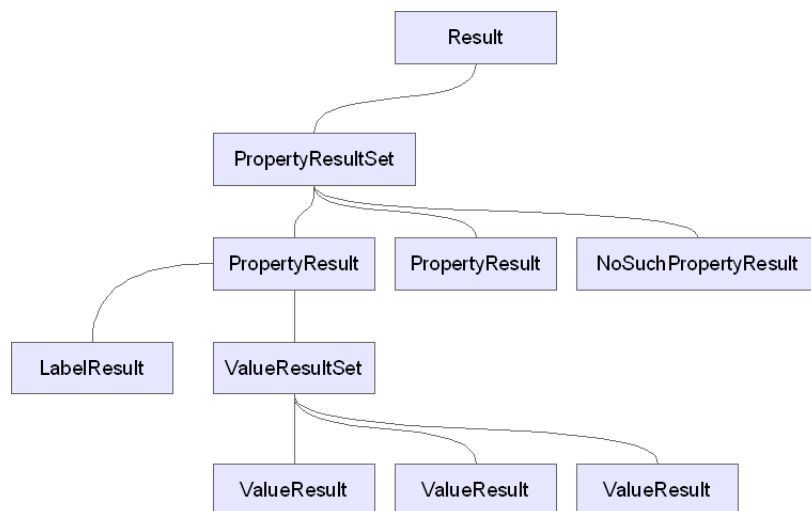


Figure 4.2. Fresnel Result Tree

²If no property could be found for a requested property, a `NoSuchPropertyResult` Object is placed in the tree (Null Object design pattern [Woo96]).

The XML output of the tree is slightly differently structured and follows the schema that can be found in Section A.1.1. An example of the output is the PIM presented in Section 7.2.3.

Chapter 5. A Visualization Ontology

This chapter describes the need for a formal definition of visualization and graphic knowledge for SemVis and subsequently introduces, a basic solution, the *Graphics.owl* ontology, that is used by the system.

5.1. Describing and Formalizing the Field of Visualization

A lot of theoretical knowledge and facts gained from experiments with the human perception are available for the field of visualization and graphic. If it is possible to formalize generally accepted facts and knowledge in shape of an ontology, this can be used as a basis for decisions by visualization systems. The graphical knowledge, once made explicit, could easily be reused by other applications, when published as part of the semantic web, in a standardized language such as RDF. Specializations and personalizations of these general facts could overwrite the general settings, if necessary.

This graphical or visualization ontology could include statements concerning the maximum number of lines in a chart that can be easily distinguished by color. And it might also include sets of colors that go well together. The independence or mutual influence of variables could also be stated in this vocabulary. As an example take the *length* and *width* of an object. These two dimensions could be given meaning and they can be varied independently in a visualization. However, the visual variable *area* is dependent on these two variables and therefore can not represent semantics along with the other two variables.

Duke et.al. [DBD04] describe the need for an ontology of visualization with regard to collaboration, (web) services, and as a basis for research and education. [ARS06] wrote a prototype ontology for visualization, building on existing yet fragmentary taxonomies such as the *Data State Model* which is algorithm oriented and several taxonomies which are data oriented.

Unfortunately these ontologies lack a definition of Visual Variables, since their focus is not on graphical knowledge, and therefore do not fulfil SemVis needs.

Because creating a complete ontology is beyond the scope of this work, this ontology only contains very basic terms, such as the Visual Variables or the properties to define which of these variables are orthogonal. But such an ontology could be extended to equip the visualization system with further graphic knowledge that is valuable, when mapping via the user interface.

5.2. Overview

The *Graphics.owl* ontology has been developed to define a vocabulary of visualization for use in SemVis. Figure 5.1 gives an overview of all classes (drawn as ellipses), properties (drawn as labels on the arrows or as attributes of the classes) and instances (drawn as rectangles). The following sections will explain the purpose of each class in detail.

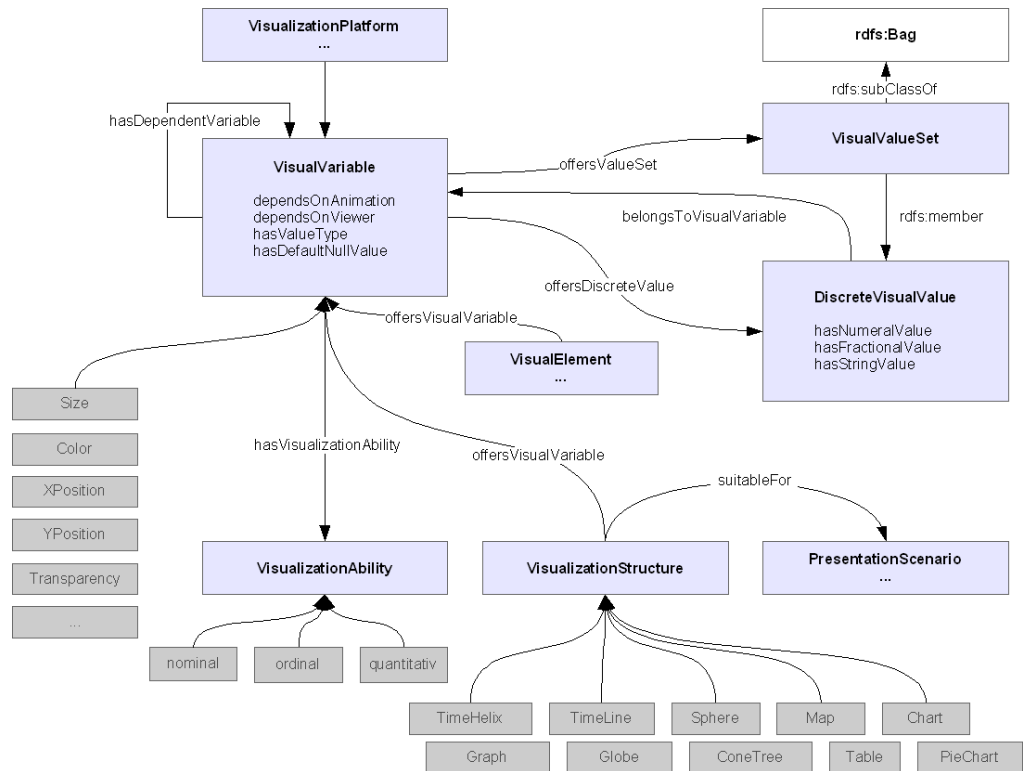


Figure 5.1. The Graphics.owl ontology — Overview

5.3. VisualVariable

Visual Variables have been described by [Ber81] and are the elements of the graphical vocabulary [Zeh04]. Bertin identifies the visual variables *area*, *size*, *brightness*, *texture*, *color*, *orientation* and *shape*, which have been formalized in the ontology. Several other variables like *depth*, *rotation* and *transparency* have been added, to fulfil the needs of other output formats than print (c.f. Figure 5.1).

```
:VisualVariable
  a      owl:Class ;
  .
```

The reflexive, symmetric property `hasDependentVariable` states that two `VisualVariables` are not fully orthogonal, but depend on each other.

```
:hasDependentVariable
  a      owl:ObjectProperty , owl:SymmetricProperty ;
  rdfs:domain :VisualVariable ;
  rdfs:range :VisualVariable ;
  owl:inverseOf :hasDependentVariable ;
  .
```

Mackinlay described and tested the value of Visual Variables for the transmission of data belonging to the three categories *nominal*, *ordinal* or *quantitative* [Mac86]. A `VisualVariables` ability to be used for data of one of these categories can be stated by the property `hasVisualizationAbility`. This allows the visualization system to automatically propose appropriate `VisualVariables`.

```
:hasVisualizationAbility
  a      owl:ObjectProperty ;
```



```

rdfs:domain :VisualVariable ;
rdfs:range :VisualizationAbility ;
.

```

Every `VisualVariable` defines a default null value that states which value carries the least meaning, but does not make the whole instance disappear. E.g. the null value for `Color` is the `DiscreteVisualValue` `grey`, however values of simple data types are also allowed.

```

:hasDefaultNullValue
  a      rdf:Property ;
  rdfs:domain :VisualVariable ;
.

```

The type of a `VisualVariables` values is defined by the property `hasValueType`.¹

```

:hasValueType
  a      rdf:Property ;
  rdfs:domain :VisualVariable ;
.

```

Other properties are the dependency of the viewpoint of the user (`dependsOnViewer`) and the dependency of animation (`dependsOnAnimation`).

```

:dependsOnViewer
  a      owl:DatatypeProperty ;
  rdfs:domain :VisualVariable ;
  rdfs:range xsd:boolean ;
.

:dependsOnAnimation
  a      owl:DatatypeProperty ;
  rdfs:domain :VisualVariable ;
  rdfs:range xsd:boolean ;
.

```

5.4. DiscreteVisualValue

`VisualVariables` can be assigned a number of possible `DiscreteVisualValues`. The property `offersDiscreteValue` is used to do that. The associated inverse property is called `belongsToVisualVariable`.

```

:DiscreteVisualValue
  a      owl:Class ;
.

:offersDiscreteValue
  a      owl:ObjectProperty ;
  rdfs:domain :VisualVariable ;
  rdfs:range :DiscreteVisualValue ;
  owl:inverseOf :belongsToVisualVariable ;
.

:belongsToVisualVariable
  a      owl:ObjectProperty ;
  rdfs:domain :DiscreteVisualValue ;
  rdfs:range :VisualVariable ;
  owl:inverseOf :offersDiscretValue ;
.

```

¹This is needed to dynamically generate values for a variable (c.f. Section 6.4)

The assignment of numeral values can be done with the property `hasNumeralValue` and `hasFractionalValue` (a sub property of `hasNumeralValue`). While `hasFractionalValue` directly assigns a value between 0 and 1, `hasNumeralValue` assigns arbitrary numbers, whose proportional value has to be dynamically calculated in relation to the range of all values that are available. The property `hasStringValue` can be used to assign a string representing the value (e.g. `#FF0000` for *Red*). The declaration of `DiscreteVisualValues` does not implicate that no other values for a `VisualVariable` exist.

```

:hasNumeralValue
  a owl:DatatypeProperty ;
  rdfs:domain :DiscreteVisualValue ;
  rdfs:range xsd:float ;
.

:hasFractionalValue
  a owl:DatatypeProperty ;
  rdfs:subPropertyOf :hasNumeralValue ;
  rdfs:domain :DiscreteVisualValue ;
  rdfs:range xsd:float ;
.

default:hasStringValue
  a owl:DatatypeProperty ;
  rdfs:domain :DiscreteVisualValue ;
  rdfs:range xsd:string ;
.

```

With the class `VisualValueSet` it is possible to define sets of `DiscreteVisualValues` that a `VisualVariable` offers. This can be useful to define colors which are a good fit, for example.

```

:VisualValueSet
  a owl:Class ;
  rdfs:subClassOf rdfs:Bag ;
.

:offersValueSet
  a owl:ObjectProperty ;
  rdfs:domain :VisualVariable ;
  rdfs:range :VisualValueSet ;
.

```

5.5. VisualElement

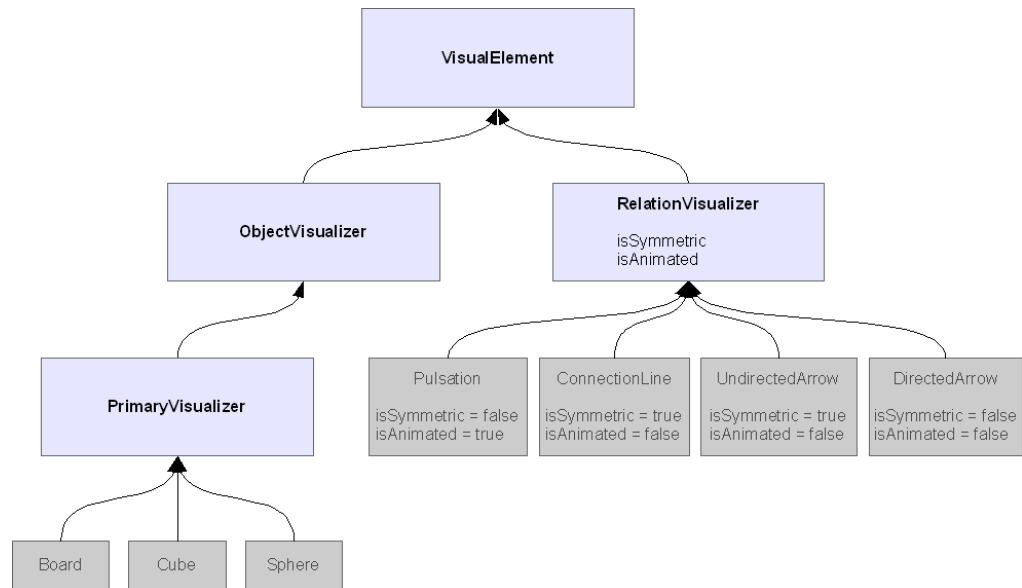


Figure 5.2. *The Graphics.owl Ontology — VisualElements*

Any object that can be seen in the visualization is a `VisualElement`. Subclasses define `ObjectVisualizers` and `RelationVisualizers` (c.f. Figure 5.2). A `PrimaryVisualizer` is a special `ObjectVisualizer` and defines all objects which can be used to visualize Primaries (c.f. Example 4.4).

```

:VisualElement
  a      owl:Class ;
.

:ObjectVisualizer
  a      owl:Class ;
  rdfs:subClassOf :VisualElement ;
.

:PrimaryVisualizer
  a      owl:Class ;
  rdfs:subClassOf :VisualElement ;
.

:RelationVisualizer
  a      owl:Class ;
  rdfs:subClassOf :ObjectVisualizer ;
.
  
```

`RelationVisualizers` can be used to explicitly show the relation between two values. They split into symmetric and asymmetric, depending on the fact if they are directed or not. `DirectedArrow` is an example for a directed, asymmetric `RelationVisualizer` whereas `UndirectedArrow` and `ConnectionLine` are examples for a symmetric one. It can also be stated if they are animated or not. Other elements for explicit relation visualization are possible [Pol06].

```

:isSymmetric
  a      owl:DatatypeProperty ;
  rdfs:domain :RelationVisualizer ;
  
```

```

    rdfs:range xsd:boolean ;
.

:isAnimated
    a          owl:DatatypeProperty ;
    rdfs:domain :RelationVisualizer ;
    rdfs:range xsd:boolean ;
.

```

For each `VisualElement` it can be stated which `VisualVariables` it supports. Although any element could theoretically have almost any variable, it should be able to discourage the use of a variable with a particular element. For example, *Transparency* should not be used with lines to carry meaning.

```

:offersVisualVariable
    a          owl:ObjectProperty ;
    rdfs:domain :VisualizationPlatform, VisualElement,
VisualizationStructure ;
    rdfs:range :VisualVariable ;
.

```

5.6. VisualizationStructure

A `VisualizationStructure` describes the characteristics of the structure that is the result of a technique of visualization, including the shape, and suitability for different presentation scenarios (`suitableFor`) but not the technique itself.

```

:VisualizationStructure
    a          owl:Class .

:suitableFor
    a          owl:ObjectProperty ;
    rdfs:domain :VisualizationStructure;
    rdfs:range :PresentationScenario .

```

Additionally, for each `VisualizationVariable`, it can be defined, if the structure supports its display (property already introduced in Section 5.3).

```

:offersVisualVariable
    a          owl:ObjectProperty ;
    rdfs:domain :VisualizationPlatform, VisualElement,
VisualizationStructure ;
    rdfs:range :VisualVariable ;
.

```

5.7. VisualizationPlatform

The class `VisualizationPlatform` represents a final output format. Each platform must be described regarding to its possibilities. Besides the available `VisualVariables`, each `VisualizationPlatform` states, if it offers `Animation`, if it offers `Interaction` and how many spatial dimensions are supported (c.f. Example 5.1).

```
:VisualizationPlatform
  a      owl:Class .

:offersAnimation
  a      owl:DatatypeProperty ;
  rdfs:domain :PresentationScenario, VisualizationPlatform;
  rdfs:range xsd:boolean .

:offersInteraction
  a      owl:DatatypeProperty ;
  rdfs:domain :PresentationScenario, VisualizationPlatform;
  rdfs:range xsd:boolean .

:offersSpatialDimensions
  a      owl:DatatypeProperty ;
  rdfs:domain :PresentationScenario, VisualizationPlatform;
  rdfs:range xsd:nonNegativeInteger .

:offersVisualVariable
  a      owl:ObjectProperty ;
  rdfs:domain :
    VisualizationPlatform,
    VisualElement,
    VisualizationStructure ;
  rdfs:range :VisualVariable ;
.
```

Example 5.1. VisualizationPlatform Class

5.8. PresentationScenario

The `PresentationScenario` describes a use case of the visualization. Here the hardware and characteristics of the construction and the probable audience can be declared. The current vocabulary allows to define, if stereo display is supported (`offersSpatialDimension`), how many colors the systems graphic devices can display the maximum resolution of the target system is, if videos can be played (`offersAnimation`) and if interaction is possible at all (`offersInteraction`). The position of the user, i.e. if the situation is immersive or not, can be stated by the property `isImmersive`.

Three of the properties are also used to describe the `VisualizationPlatform`: `offersSpatialDimensions`, `offersAnimation` and `offersInteraction` have both classes as their domain (c.f. Example 5.2).

```
:PresentationScenario
  a      owl:Class .

# shared with VisualizationPlatform:
:offersSpatialDimensions
  a      owl:DatatypeProperty ;
  rdfs:domain :PresentationScenario, VisualizationPlatform;
  rdfs:range xsd:int .

:offersColor
  a      owl:DatatypeProperty ;
  rdfs:domain :PresentationScenario ;
  rdfs:range xsd:int .

:offersResolution
  a      owl:DatatypeProperty ;
  rdfs:domain :PresentationScenario ;
  rdfs:range xsd:int .

# shared with VisualizationPlatform:
:offersAnimation
  a      owl:DatatypeProperty ;
  rdfs:domain :PresentationScenario, VisualizationPlatform;
  rdfs:range xsd:boolean .

# shared with VisualizationPlatform:
:offersInteraction
  a      owl:DatatypeProperty ;
  rdfs:domain :PresentationScenario, VisualizationPlatform;
  rdfs:range xsd:boolean .

:isImmersive
  a      owl:DatatypeProperty ;
  rdfs:domain :PresentationScenario ;
  rdfs:range xsd:boolean .
```

Example 5.2. PresentationScenario Class

5.9. Facts

Besides the visualization vocabulary, the *Graphics.owl* ontology additionally contains facts of the field of visualization. *VisualVariables* such as *Size*, *X-Pos*, *Y-Pos*, *Z-Pos*, *Color*, *Transparency*, *Orientation* etc. are described in regard to their ability of visualizing *nominal*, *ordinal* or *quantitative* data and also in regard to their mutual dependency. Furthermore a basis of *DiscreteVisualValues* such as *small*, *middle*, *big* for *Size* or *red*, *green*, *blue* for *Colors* is defined. Also instances of common *VisualizationStructures* such as *TimeLine*, *Table* etc. are described. Here it has to be considered, if these should be rather described as classes with appropriate restrictions and if all these facts should be kept separately from the vocabulary.

Chapter 6. A Novel Mapping Vocabulary for Semantic Visualization

For visualization purposes, a means of defining a mapping from facets to Visual Variables is needed. This mapping definition should be reusable, platform independent and allow for fine configuration as well as for a maximum of automatism.

The Fresnel display vocabulary, described in the Chapter 4, covers the styling of instances and relations with its *Format* part. However, it is not appropriate for the mapping from facets to Visual Variables. The needed mapping functionality is similar, but still different from the one of Fresnel. Fresnel offers hooks for CSS styling classes, but these are intended to decorate display elements with concrete parameter settings for the color, line width or size of all properties or instances of a certain class. The mapping vocabulary should be able to assign properties to *Visual Variables*, as defined in the *Graphics.owl* ontology in Section 5.1.

The Fresnel style information, resulting in the `class`-attributes of the result tree, could have been used for the mapping, but this would have been a misappropriation. Even if the style information was used, a lot of extensions would have been required, in order to add advanced mapping features.

Therefore, we developed a new basic vocabulary for the mapping, orientated at Fresnel. It is, as is Fresnel, fully declarative and written in OWL. This allows for a consistent editing of the definition files and integration of the two languages. The definition of the mapping in an extra file makes it possible to quickly exchange a mapping and define alternative mappings for special purposes.

This chapter will explain the challenges for a flexible mapping in general, along with a description of the single constructs in the new mapping vocabulary, which is defined by the *Mapping.owl* ontology. Short examples of their use in a mapping definition file can be found at the end of each section.

6.1. Overview

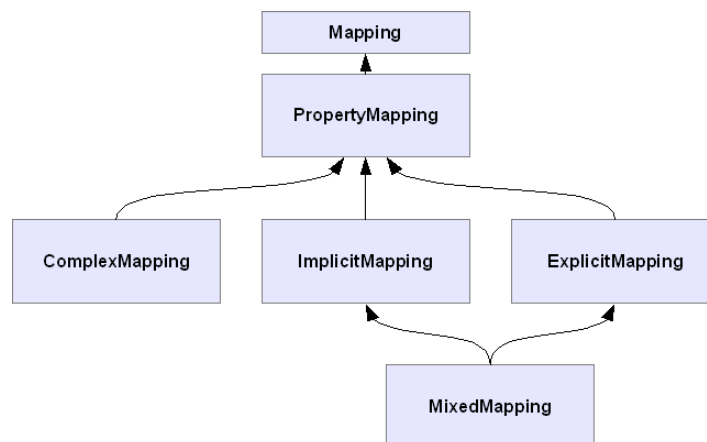


Figure 6.1. Subclasses of Mapping

As Figure 6.1 shows, the mapping from facets to Visual Variables can be subdivided into implicit, explicit and complex `PropertyMappings`. An `ImplicitMapping`

can be configured using several parameters. It then calculates the values of the Visual Variable for a concrete facet value based on these parameters. By contrast, an `ExplicitMapping` allows for manually mapping single values to each other. The best from both worlds can be found in the `MixedMapping`. It can be configured to implicitly map the values, however, if there is need, single values can be assigned manually. Complex structures of a visualization can be described with a `ComplexMapping` which is able to process more than one Visual Variable and can consider the context of a value.

Figure 6.2 gives an overview of the complete mapping vocabulary, together with example instances (darkgrey rectangles). The following sections describe the elements of the language in detail.

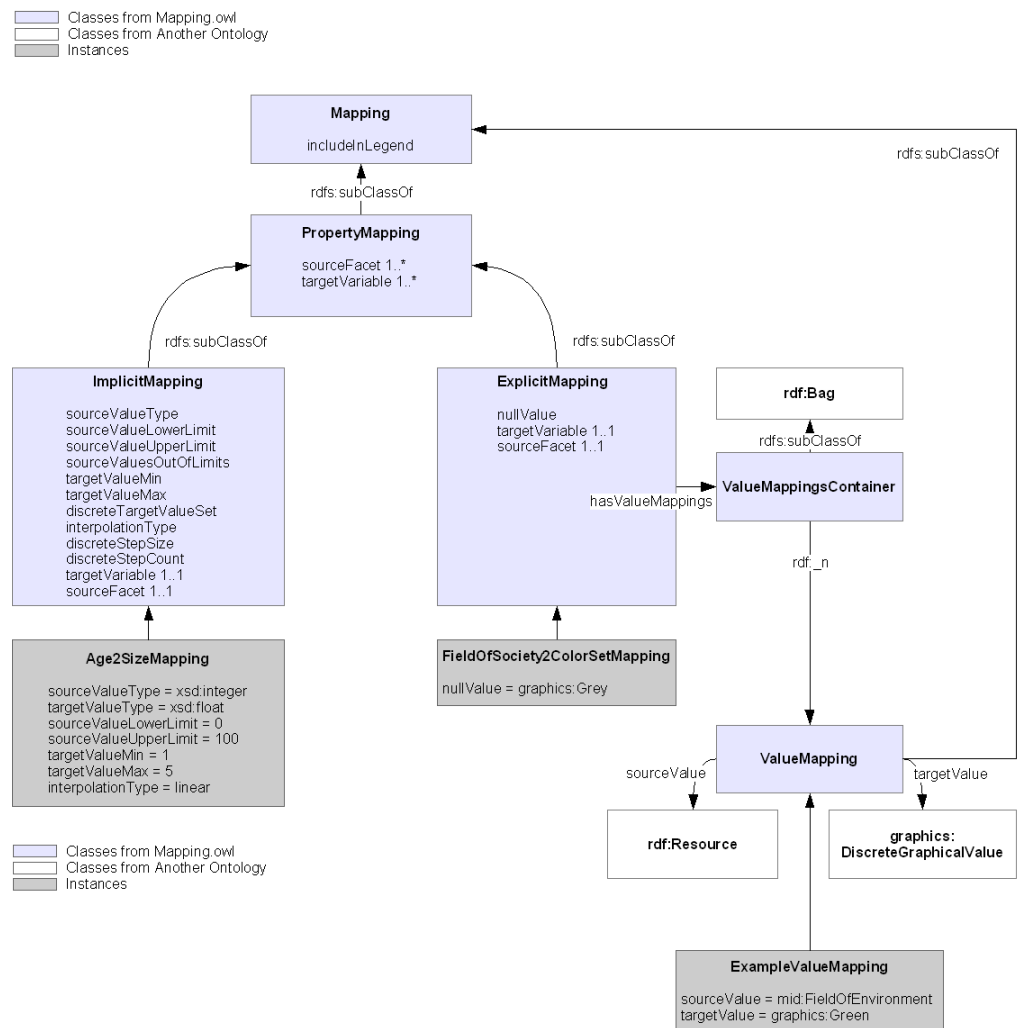


Figure 6.2. Mapping.owl

6.2. Mapping

The super class of all mappings is the class `Mapping`. A first, but abandoned idea was to define the mapping by a single statement, using the property `map:isMappedTo`:

```
facetA isMappedTo visualVariableB .
```

However, this turned out to be insufficient for defining details of a mapping and was not consistent with the way Fresnel works. In terms of clean ontology construction, this is even semantically wrong, since the fact of being mapped to `visualVariableB` is not

generally true for *facetA*, but applies only in a certain application for a certain purpose. Instead of using a simple statement, the mapping is now modelled as an instance of the Mapping class.

Common to all Mappings, is the option of being included in a legend or not. For this purpose, all Mappings share the boolean property `includeInLegend`.

```
:Mapping
  a owl:Class ;
  .

:includeInLegend
  a owl:DatatypeProperty ;
  rdfs:range xsd:boolean ;
  rdfs:domain :Mapping ;
  .
```

6.3. PropertyMapping

A `PropertyMapping` defines the mapping from one facet to a `VisualVariable`. However, this only draws a general connection between a facet and a `VisualVariable`. The interesting part remains the mapping of the values, which is left to the subclasses of `PropertyMapping`.

```
:PropertyMapping
  a owl:Class ;
  rdfs:subClassOf :Mapping;
  rdfs:subClassOf [
    a owl:Restriction ;
    owl:onProperty :sourceFacet ;
    owl:minCardinality "1"^^xsd:nonNegativeInteger ;
  ]
  rdfs:subClassOf [
    a owl:Restriction ;
    owl:onProperty :targetVariable ;
    owl:minCardinality "1"^^xsd:nonNegativeInteger ;
  ]
  .
```

The properties `sourceFacet` and `targetVariable`¹ must be defined for each `PropertyMapping` and therefore the minimum cardinality for these two properties is set to 1. The restriction of the maximum cardinality of `sourceFacet` and `targetVariable` is left to the subclasses of `PropertyMapping`.

6.3.1. sourceFacet

The property `sourceFacet` points to the source property of the mapping. The range is *rdf:Property*.

```
:sourceFacet
  a owl:ObjectProperty ;
  rdfs:range rdf:Property ;
  rdfs:domain :PropertyMapping ;
  .
```

6.3.2. targetVariable

The required property `targetVariable` points to the target `graphics:VisualVariables` of the mapping. There may be more than

¹I avoided to use the terms range and domain in this context, to avoid confusion with the domain and range of properties used in RDFS.

one `targetVariable` defined with `ComplexMappings`. All the defined `graphics:VisualVariables` will be affected by the mapping.

```
:targetVariable
a owl:ObjectProperty ;
rdfs:range graphics:VisualVariable ;
rdfs:domain :PropertyMapping ;
.
```

`PropertyMapping` has three specializations: `ImplicitMapping`, `ExplicitMapping` and `MixedMapping` which are now further described.

6.4. ImplicitMapping

An `ImplicitMapping` is a `PropertyMapping` which implicitly maps the facets values to `VisualVariables` and does not require an explicit mapping for each value. When performing an `ImplicitMapping`, `SemVis` takes all source values within defined limits and maps them to the whole available range of `Visual Variable` values, which is defined by a minimum and maximum. An `ImplicitMapping` must have exactly one `sourceFacet` and `targetVariable`.

```
:ImplicitMapping
a owl:Class ;
rdfs:subClassOf :Mapping;
rdfs:subClassOf [
a owl:Restriction ;
owl:onProperty :targetVariable ;
owl:cardinality "1"^^xsd:nonNegativeInteger ;
]
rdfs:subClassOf [
a owl:Restriction ;
owl:onProperty :sourceFacet ;
owl:cardinality "1"^^xsd:nonNegativeInteger ;
]
.
```

To configure the mapping, several parameters such as the limits for source values and target values, as well as the interpolation type can be defined by properties of the `ImplicitMapping`. Section 6.4.3 explains these properties that are common to all implicit mappings.

Depending on the characteristics of the source and target values it can be distinguished between variants of `ImplicitMappings` (c.f Figure 6.3). Section 6.4.4 and the following sections describe the four possible combinations of mappings from continuous to continuous, continuous to discrete, discrete to discrete and discrete to continuous. Additional properties that are required for a particular variant are introduced at the end of each section.

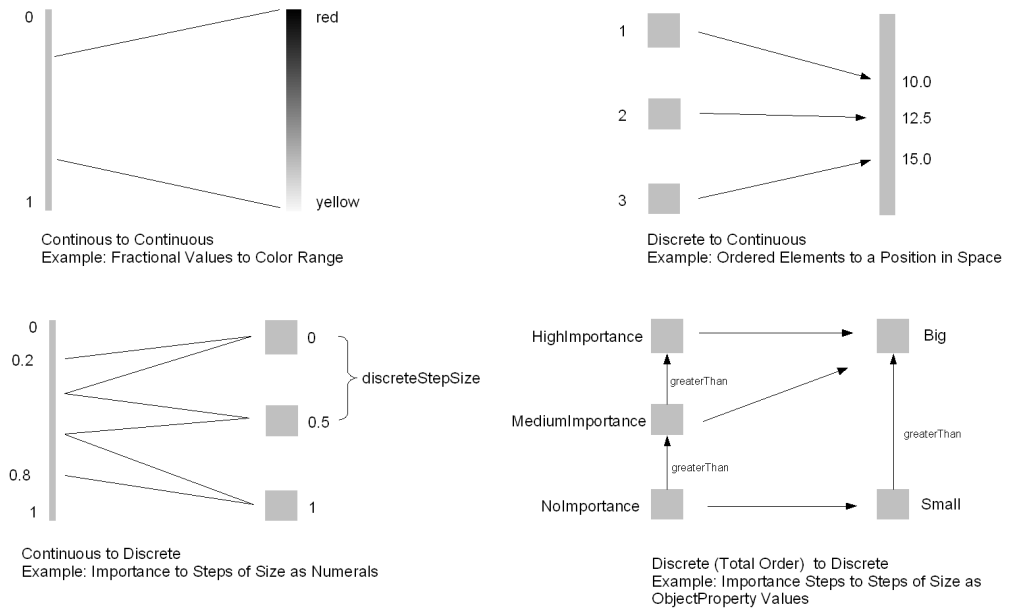


Figure 6.3. Mappings between Discrete and Continuous Values

6.4.1. Relations between ObjectProperty Values

Depending on the existing in-between-relationships, the data can be classified as *nominal*, *ordinal* or *quantitative* (also c.f. Section 2.3.3). As described in Section 5.3, the `VisualVariables` are equipped with a property to state their ability for each of these characteristics of the data. But the relations between the values can be used to also allow the implicit mapping of discrete values of `owl:ObjectProperty`s. This applies equally to value sets on the source and the target side of the mapping.

While all `ObjectProperty` values are nominal data, some of them do not offer any order relation and are consequently not ordinal. If, however, such an order relation exists, it can further be distinguished between a total order (e.g. forming a list structure) and a partial order (e.g. forming a tree structure). See Figure 6.4 for an example of tree structured data that is mapped to color values. The similarity of the colors is based on the structural distance of two nodes in the tree.

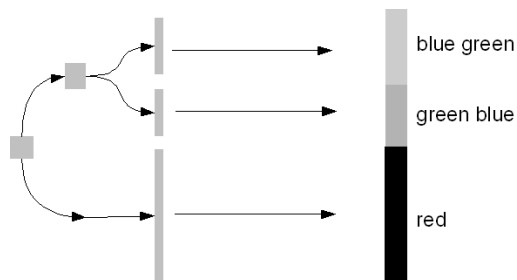


Figure 6.4. Mapping Ordinal Data (Tree Structure)

An example for the more usual case of a list structure was already given in Figure 6.3 (bottom, right), where the discrete source values are ordered by the relation `greaterThan`.

Quantitative data offers information on the proportion between two values. This applies to all numeral values, but is difficult to express for `ObjectProperty` values in RDF due

to the fact that this can not be expressed directly by a binary relation. However, design patterns for the description of n-ary relations exist, [N-Ary].

6.4.2. Adding Order and Ratio to ObjectProperty Values

Alternatively to the definition of a new order relation between *ObjectProperty* values and connecting them by this means, a numeral value can be attached, which is able to provide the ordinal information and additionally also quantitative information. The properties `graphics:hasNumeralValue` and `graphics:hasFractionalValue` can be reused for this purpose (c.f. Section 5.4).

6.4.3. Common properties

These properties are common to all `ImplicitMappings`.

Value Types

To interpret the source values correctly and to be able to generate correct target values, the type of the source and target values has to be known to the system. The type of the target variable values is given by the `graphics:hasValueType` property of the `graphics:VisualVariable`. The type of the values for the source facet does not need to be the same for all values. Since that, it has to be stated, which type is handled by the mapping.

```
:sourceValueDatatype
  a owl:ObjectProperty ;
  rdfs:domain :ImplicitMapping;
.
```

Source Value Limits

The setting of limits for the source values is optional. The properties `sourceValueLowerLimit` and `sourceValueUpperLimit` define the span of source values which are mapped. Other values, outside this span, are either cut² or set to the limiting values, depending on the value for the property `sourceValuesOutOfLimits`. If no limits are defined, the range of source values should be calculated from all available source values.

```
:sourceValueLowerLimit
  a rdf:Property ;
  rdfs:domain :ImplicitMapping;
.

:sourceValueUpperLimit
  a rdf:Property ;
  rdfs:domain :ImplicitMapping;
.

:sourceValueOutOfLimits
  a owl:DatatypeProperty ;
  rdfs:range [
    a owl:DataRange ;
    owl:oneOf ("limits"^^xsd:string "cut"^^xsd:string)
  ] ;
  rdfs:domain :ImplicitMapping;
.
```

²This equals a selection on a higher processing level.

Target Value Range

The properties `targetValueMin` and `targetValueMax` are required and define a minimum and maximum value for the `graphics:VisualValues`. The range of the two properties could be restricted to equal the type of the target variables values.

```
:targetValueMin
a rdf:Property ;
rdfs:domain :ImplicitMapping;
.

:targetValueMax
a rdf:Property ;
rdfs:domain :ImplicitMapping;
.
```

Interpolation

An `ImplicitMapping` needs to define, how the interpolation of values is done. The property `interpolationType` is used for that and can have the values *linear*, *logarithmic* or *exponential*³. The default value should be *linear*.

```
:interpolationType
a owl:DatatypeProperty ;
rdfs:range [
a owl:DataRange ;
owl:oneOf (
"linear"^^xsd:string
"logarithmic"^^xsd:string
"exponentiell"^^xsd:string
)
] ;
rdfs:domain :ImplicitMapping;
.
```

6.4.4. Continuous to Continuous

The most simplistic case is the mapping of continuous facets to continuous Visual Variable values. This can be configured by the common properties already introduced before. Figure 6.3 (top, left) shows an example of this kind of mapping. Here fractional values, which could represent the importance of something (expressed as numerals, ranging from 0 to 1), are mapped to a color gradient that is described by using the properties `targetValueMin` (set to `graphics:Red`) and `targetValueMax` (set to `graphics:Yellow`). Please note that not the whole range of potential source values is chosen, but the source values are limited. Only the remaining range is mapped, where the lower limit value corresponds to yellow and the upper limit value to red. Other values can be set to the limiting values, for example. This behavior can be achieved by setting the property `sourceValuesOutOfLimits` to *limits* instead of the default value *cut*.

6.4.5. Continuous to Discrete

A mapping from a continuous facet to discrete visual values can also easily be performed. Figure 6.3 (bottom, left) shows how intervals of values are mapped to discrete numeral values. For example, again values from the facet `importance` (ranging between 0.0 and 1.0) can be mapped to the discrete numeral values of the `Visual-Variable Size` with the fixed sizes *0.0*, *0.5*, and *1.0*.

³This property needs more parameters to be configured and should make use of [MathML].

However, the discrete values on the target side can also be *ObjectProperty* values. In this case the definition of an order relation between the values, as described in Section 6.4.1, is required. If only the mutual distinction of values is the goal of a mapping, but no information on the relation between the source values shall be expressed, the source values can be mapped arbitrarily to the discrete or quantized target values.

discreteStepSize and discreteStepCount

The step size can be set with the property `discreteStepSize` or, alternatively, with the property `discreteStepCount` in addition to the required maximum and minimum values for the target variable.

```
:discreteStepSize
  a owl:DatatypeProperty ;
  rdfs:range xsd:float ;
  rdfs:domain :ImplicitMapping;
.

:discreteStepCount
  a owl:DatatypeProperty ;
  rdfs:range xsd:nonNegativeInteger ;
  rdfs:domain :ImplicitMapping;
.
```

Set of Target Values

The set of objects that are used as values for the mapping can be optionally limited with the explicit definition of a `graphics:VisualValueSet`. As an example such a set could define a number of colors that match well. Predefined sets are provided as facts in the *Graphics.owl* ontology. A set of `DiscreteVisualValues` can be defined with the property `discreteTargetValueSet`.

```
:discreteTargetValueSet
  a owl:ObjectProperty ;
  rdfs:range graphics:VisualValueSet ;
  rdfs:domain :ImplicitMapping;
.
```

The existence of a `TargetValueSet` should be required by restrictions, if no `targetValueMin` and `targetValueMax` values are given.

6.4.6. Discrete to Continuous

The other way around also the mapping from discrete *ObjectProperty* values or numerals to continuous values has to be possible. Again providing an order between the object values is required. Figure 6.3 (top, right) shows, how object values, that form a list are mapped to values within a range from *10.0* to *15.0*. An example for this is the positioning of sorted items in space. If quantitative values are offered (c.f. Section 6.4.1), the ratio of the source values can be mapped to the target values. If only ordinal values are provided, as in the example, the target values will all have equal distance (for a linear mapping).

6.4.7. Discrete to Discrete

For this kind of mapping, the problems of order and quantitative information for discrete values apply to the source and target side of the mapping. Figure 6.3 (bottom, right) shows, how the object values labeled *NoImportance*, *MediumImportance*, and *HighImportance* are implicitly mapped to `DiscreteVisualValues` from the *VisualVariable Size*. For this example, consider the values of the facet and the Vi-

sualVariable as related to each other by the `greaterThan` relationship. This forms two lists, which can be implicitly mapped.

Again, the unproportional mapping of values, requires quantitative information.

6.5. ExplicitMapping

With the help of an `ExplicitMapping`, values can be manually mapped to each other. This is often necessary for discrete values of `ObjectProperty`s which lack any structure between them (either on the source or target side), so that no implicit mapping can be performed. But an explicit manual mapping might also be needed for `owl:DatatypeProperty`s with integers or strings as a range, whenever a precise value assignment is required. See Figure 6.5 for an example.

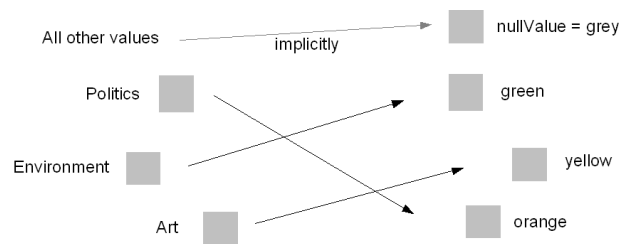


Figure 6.5. Explicit Mapping of Discrete Unstructured Values

```

:ExplicitMapping
  a owl:Class ;
  rdfs:subClassOf :Mapping ;
  rdfs:subClassOf [
    a owl:Restriction ;
    owl:onProperty :targetVariable ;
    owl:cardinality "1"^^xsd:nonNegativeInteger ;
  ]
  rdfs:subClassOf [
    a owl:Restriction ;
    owl:onProperty :sourceFacet ;
    owl:cardinality "1"^^xsd:nonNegativeInteger ;
  ]
.

```

In the following, the constructs are described that are necessary for the mapping of the single values and also, how it is proceeded, if a value is missing.

6.5.1. ValueMapping

An instance of `ValueMapping` maps a discrete facet value to a discrete Visual Variable value. `ValueMapping` is a subclass of `Mapping` and inherits the property `includeInLegend`. This way, it can consistently be defined, if this `ValueMapping` should appear in a legend.

```

:ValueMapping
  a owl:Class ;
  rdfs:subClassOf :Mapping ;
.

```

The following two properties are used with a `ValueMapping` and define the source and target value.⁴

⁴Currently `targetValue` allows only `graphics:DiscreteVisualValues` to be the target value. This should be relaxed.

```

:sourceValue
  a rdf:Property ;
  rdfs:domain :ValueMapping ;
  rdfs:range rdf:Resource ;
.

:targetValue
  a owl:ObjectProperty ;
  rdfs:domain :ValueMapping ;
  rdfs:range graphics:DiscreteVisualValue ;
.

```

6.5.2. ValueMappingsContainer

`ValueMappingsContainer` is not a `Mapping` itself, but a class that works as a container for all instances of `ValueMapping` belonging to an `ExplicitMapping`. It extends the *rdf:Bag* class.

```

:ValueMappings
  a owl:Class ;
  rdfs:subClassOf rdf:Bag;
.

```

The container providing the set of `ValueMapping` instances is attached to an `ExplicitMapping` with the required property `hasValueMappings`.

```

:hasValueMappings
  a owl:ObjectProperty ;
  rdfs:domain :ExplicitMapping ;
  rdfs:range :ValueMappingsContainer ;
.

```

6.5.3. nullValue

The optional property `nullValue` points to a `graphics:DiscreteVisualValue` that should be used, if no `ValueMapping` exists for a given source value. It is not intended to define the smallest value for a variable used in the visualization (this is defined by the `targetValueMin` property), but the value that carries the least possible semantics. This is usually an average value, such as *grey* in the field of colors.

If this property is undefined, the null value defaults to the `graphics:VisualVariables` `graphics:hasDefaultNullValue`. If this is also missing, a warning should be issued and the average value of all `DiscreteVisualValues` should be calculated. The null value overwrites the `VisualVariables` default null value, to allow different null values for different presentations.

```

:nullValue
  a owl:ObjectProperty ;
  rdfs:domain :ExplicitMapping ;
  rdfs:range graphics:DiscreteVisualValue ;
.

```

6.6. MixedMapping

`MixedMapping` combines implicit and explicit mappings by inheriting from both classes. When evaluating the mapping, a value from the `ExplicitMapping` overwrites values from the `ImplicitMapping`. This allows to roughly configure a mapping implicitly, but still partially define special values by hand.

```

:MixedMapping

```



```

a owl:Class ;
rdfs:subClassOf :ImplicitMapping, :ExplicitMapping ;
.

```

After having introduced the straightforward implicit and explicit mappings, the next section will deal with more advanced `PropertyMappings`.

6.7. ComplexMapping

The process of building a visualization structure can be seen as a complex mapping. This allows for a consistent handling of the construction of visualization structures and simple mappings. `ComplexMapping` is a subclass of `PropertyMapping` (c.f. Figure 6.6).

```

:ComplexMapping
a owl:Class ;
rdfs:subClassOf :PropertyMapping ;
.

```

It is attributed as complex, because the facet values can result in values for multiple `VisualVariables` (`targetVariable` is not restricted to I). Moreover, the values might be dependent on the context. As an example, take a mapping to spatial variables. The final value for the position depends on other values of neighbour objects, e.g. objects might need to be rearranged to avoid overlapping. Similarly the color value of an item might depend on color values of adjacent items to ensure a necessary level of contrast for the distinction of two values.

`ComplexMappings` are not even limited to one `sourceFacet` (the cardinality is not restricted to I). This becomes necessary with the construction of periods in timelines, for example. Here values from a `startTime` and `endTime` facet need to be processed.

A `ComplexMapping` is also responsible for providing explicit values in the form of scales. This is because a scale should be drawn dependently of the actually occurring values and must be positioned according to the structure itself.

Subclasses of `ComplexMapping` (c.f. Figure 6.6, `TimeHelixMapping`) can be used to describe a class of mappings having certain requirements, e.g. the need for particular `sourceFacets` or other characteristics, e.g. the `VisualVariables` that are affected by the mapping.

While it can be stated relatively easily that a subclass of `ComplexMappings` has certain `VisualVariables` as `targetVariables`, the restriction of `sourceFacet` values is more difficult. The solution, also displayed in the diagram as pseudo code, proposes the restriction of the `sourceFacet` to a specific facet. An instance of the mapping could meet this requirement by making its `sourceFacet` a `subPropertyOf` the required `sourceFacet`. Inference allows it to be used instead of the defined property.

However, it has to be found a way to restrict more than one `sourceFacet`. Also, a restriction to types of a facet values, rather than to specific facets, is desirable. E.g. all properties with a range of `xsd:date`, `xsd:dateTime` or `xsd:time`. This way every facet that offers time values could be assigned to the visualization structure. One possible solution is presented in Section 6.7.3.

The definition of `ComplexMapping` requires more investigation, also in regard to the possibilities of staying in OWL DL, since using a property as a value is only possible in OWL FULL.

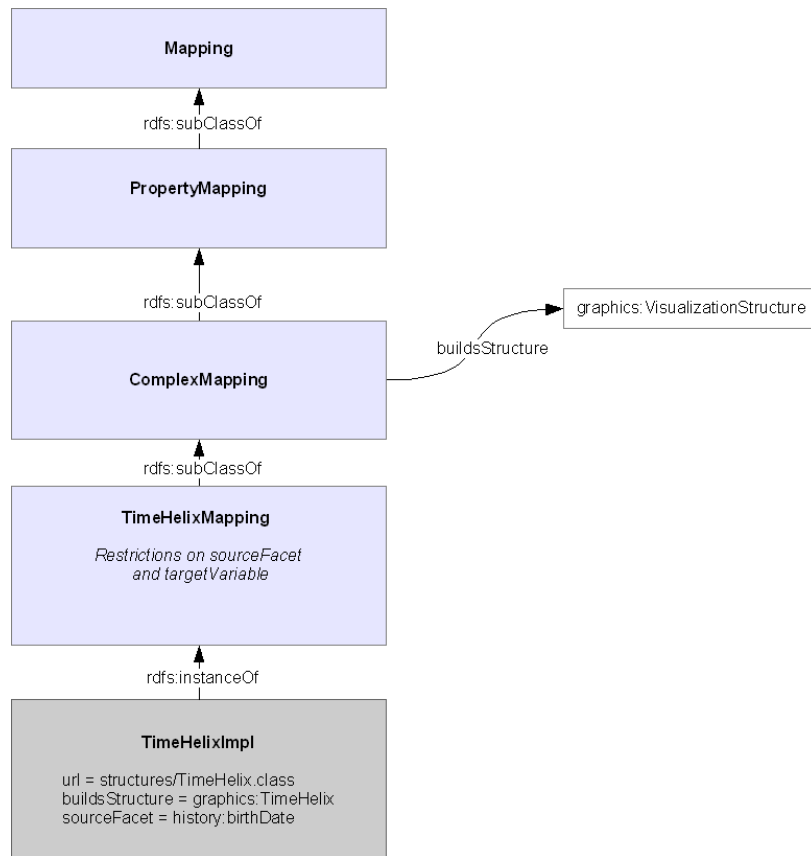


Figure 6.6. ComplexMapping

6.7.1. url

One question that occurred, when thinking about ComplexMappings, was the best place for defining the algorithms that build the structures. As these are operational semantics at the moment, they cannot be defined directly in the mapping definition files. Although the description of the algorithms with a rule language could be possible, the adoption of a non-imperative rule language was beyond the scope of this work. We decided to put the information on the construction of a visualization structure into Java classes, which are referred to by the other mapping constructs, being aware of the lack of reusability in comparison to the other descriptions. This means, when introducing a new ComplexMapping to the system, a Java class, extending the class ComplexMapping, has to be written.

The class that provides a notation of the algorithms, which perform the ComplexMapping is defined with the property `url`.

```

:url
a owl:DatatypeProperty ;
rdfs:range xsd:anyUri ;
rdfs:domain :ComplexMapping;
.

```

6.7.2. buildsStructure

This property points to an instance of VisualizationStructure as defined in the *Graphics.owl* ontology. The ComplexMapping defines its requirements and configuration possibilities from the construction point of view. However, facts on the charac-

teristics and abilities of a particular visualization structure are in the focus of the class `graphics:VisualizationStructures`. A `ComplexMapping` can be linked to an instance of this class with the property `buildsStructure`.⁵

```
:buildsStructure
  a owl:ObjectProperty ;
  rdfs:range graphics:VisualizationStructure ;
  rdfs:domain :ComplexMapping ;
.
```

6.7.3. Example

The following listing gives an example of a `ComplexMapping`, including a possible definition of a subclass that restricts its `sourceFacet` values to selected types. In the example an instance of a class called `TimeHelixMapping` is defined. This class is restricted to allow only properties with the range `xsd:dateTime`, `xsd:date` and `xsd:time` as values. Here a solution employing a meta property class is chosen.

```
# property meta class
# (globally restricting the range to selected classes)
map:TimeProperty
  a owl:Class ;
  owl:equivalentClass
    [ a owl:Class ;
      owl:intersectionOf (
        [ a rdf:Property;
          rdfs:range [
            a owl:Class ;
            owl:unionOf (
              xsd:dateTime
              xsd:time
              xsd:date )
            ]
          ]
        ]
      )
    ]
.
```

```
# (locally restriction of the TimeHelixMapping sourceFacet to
# propertys that are instances of the above defined meta class)
:TimeHelixMapping
  a owl:Class
  rdfs:subClassOf map:ComplexMapping;
  rdfs:subClassOf
    [ a owl:Restriction ;
      owl:someValuesFrom map:TimeProperty ;
      owl:onProperty map:sourceFacet
    ]
.
```

```
:aTimeHelixMapping
  a map:TimeHelixMapping ;
  :buildsStructure graphics:3DHelix ;
  :sourceFacet hist:hasStartTime ;
  :sourceFacet hist:hasEndTime ;
.
```

⁵It has to be further investigated, if the separation of `VisualizationStructure` and `ComplexMapping` is justified or if they should be merged.

6.8. Inference

When mapping data from different sources, the names of the properties to be mapped are unlikely to be the same. This would require a central global naming instance and contradict the idea of the web [OWL]. But instead of adding a mapping for every new similar, but differently named property, the ontologies can be mapped and connected to each other. This approach also improves the interconnectedness of vocabularies. Two possibilities exist to map properties from one vocabulary to another: `rdfs:subPropertyOf` and `owl:equivalentPropertyOf`.

With the help of the `rdfs:subPropertyOf` relationship, it can be stated that a property is a specialization of another more general property. This means for a property, that is mapped to a Visual Variable, that also every sub property is mapped to this Visual Variable.

However, using `rdfs:subPropertyOf` relationships is inappropriate in the case of total equivalence of two properties. When it is sufficient for an instance to be related by a property *A* to also be related to a property *B*, and not only necessary, the properties should be related with the OWL construct `owl:equivalentProperty`⁶ (c.f. Example 6.1). The mapping defines this inference by default, however it depends on the reasoner which is used, if the facts are evaluated. While `rdfs:subPropertyOf` only requires a fast RDF(S) reasoner, `owl:equivalentProperty` necessitates the use of a computational more expensive OWL reasoner. The inference behavior can not be turned off for the moment. However, an equivalent approach as in the *Fresnel Selector Language* [FSL] could be chosen, where the inference is switched on by adding a special character in front of the statement.

```
# relating properties with rdfs:subPropertyOf
presidents:born rdfs:subPropertyOf hist:hasStartTime.

# relating properties with owl:equivalentProperty
hist:hasStartTime owl:equivalentProperty someMidLevelOntology:begins.
```

Example 6.1. Ontology Mapping by Inference

6.9. Explicit Display of Relations

Relations, i.e. to *ObjectProperty* values, can be displayed explicitly. Usually, the fact that there is a connection between two instances is visualized implicitly by spatial neighbourhood. That means child instances are displayed within a `PrimaryVisualizer` or attached to it so that they can be visually associated with their parent. The Fresnel sublens mechanism supports this by allowing to choose varying representations for child instances.

However, there might be situations, when more than one class is defined as a *Primary*, or there might exist a reflexive relationship for a *Primary*. Then it is useful no to display additional instances as children, but to display the instances where they are positioned by the mapping and show the relation between a pair of instances with the help of additional graphical elements. A standard means for this is the use of arrows or connection lines depending on the directedness of the relation. These are described as `RelationVisualizers` in the *Graphics.owl* ontology (see also Section 5.1).

⁶`owl:sameAs`, which states equality between individuals, could also be used to map properties. However, since this requires treating properties as individuals, this is only allowed in OWL FULL, which has expensive reasoning characteristics. C.f. the definition of `equivalentProperty`: <http://www.w3.org/TR/owl-ref/#equivalentProperty-def>

The definition of explicit relation visualization can be done in SemVis in the same way that properties are mapped to Visual Variables: The `targetVariable`⁷ of a `PropertyMapping` might also point to a `RelationVisualizer`. `RelationVisualizers` may be used in combination with a mapping to `VisualVariables`.

6.10. Limitations

In contrast to Fresnel, the SemVis mapping is unaware of the context at the moment. A property is always mapped to the same Visual Variable. However, there might be situations, where, depending on the values of other assigned Visual Variables, such as position in space, different graphical properties have to be employed. At the current state a mapping is equally valid for the whole visualization and every *Primary* whereas Fresnel allows for the definition of `Formats` for properties depending on the instance that uses them. Only in `ComplexMappings` it is possible to map multiple properties to the same Visual Variable.

⁷It could be argued that the property `targetVariable` should be renamed to allow this generalization. It could be changed to `target`.

Chapter 7. A Model-Driven Architecture for Flexible Visualization

The system architecture of SemVis follows the principles of a *Model-Driven Architecture* (MDA) as described by the *Object Modelling Group* (OMG). A Model-Driven Architecture consists of *Platform Independent Models* (PIM) that are transformed, mostly with the help of additional knowledge, to more specific models, the *Platform Specific Models* (PSM) and finally into code, representing the most specific model. A model can only be referred to as *platform specific* or *platform independent* when seen from a certain viewpoint, since the terms are relative. This viewpoint is the final Visualization Platform throughout this chapter.

The reason for the model-driven approach was the need for a maximum of variability. The system should be variable in two ways (c.f. Figure 7.1): At first the Visualization Platform should be exchangeable, and secondly the mapping of facets to Visual Variables should be exchangeable as well. Figure 7.1 shows the two variation points. The first is the mapping from facets (on the left side) to a general concept of a *Visual Variable* (center), which is formalized by the *VisualVariables* of the *Graphics.owl* vocabulary. In a second step, these general concepts have to be transformed to means that the final Visualization Platform offers (right side). Some concepts might not be applicable to a particular *Visualization Platform*. In this case the system creates an alternative representation.

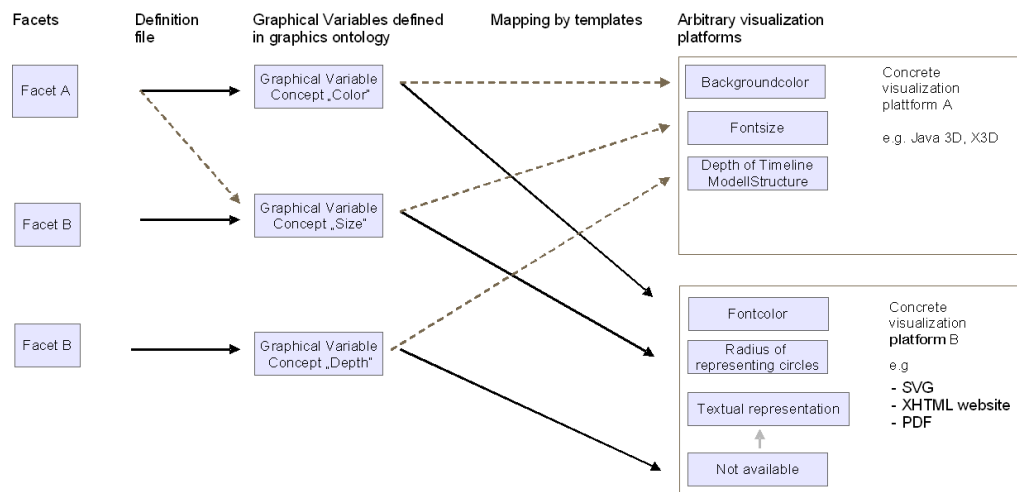


Figure 7.1. Variability of SemVis

The structure of this chapter is as follows: Section 7.1 shows what the model transformation steps in SemVis are, which additional knowledge is added at what time of the overall process and which PIMs and PSMs are generated. Section 7.4 describes this additional knowledge and Section 7.2 describes each application of the MDA pattern in detail. Finally, Section 7.5 compares SemVis to the *Graphical Modeling Framework for Eclipse* (GMF).

7.1. A Model-Driven Architecture

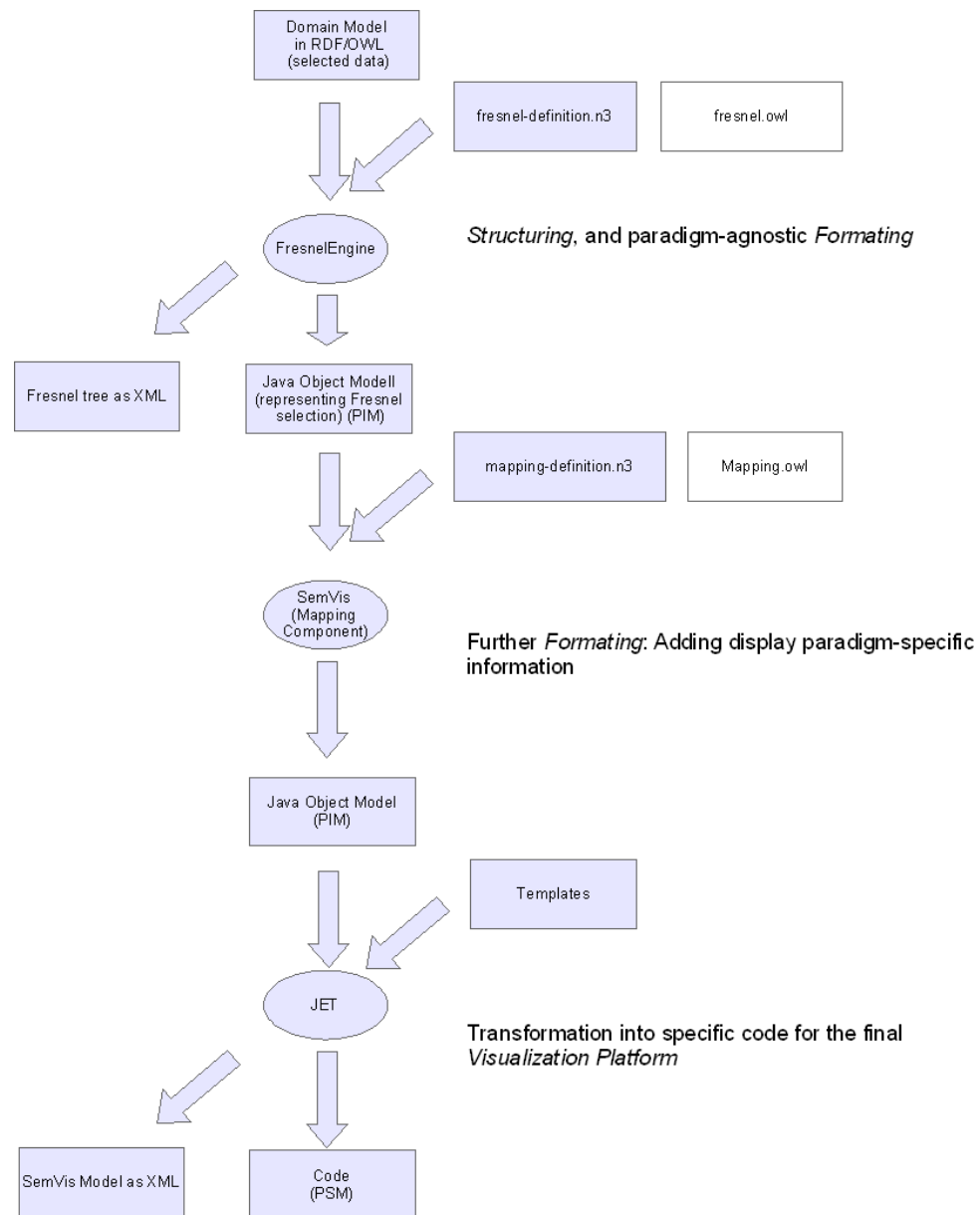


Figure 7.2. *SemVis as a Model-Driven Architecture*

Figure 7.2 shows the architecture of SemVis from the point of the *Model-Driven Architecture*. The SemVis architecture performs three transformation steps:

The first transformation is done by the integrated Fresnel Engine. It uses the presentation knowledge in the *fresnel-definition.n3* file for selecting and formatting the RDF data, which represents the *Domain Model*. The added presentation knowledge is fully presentation-paradigm-agnostic. There is the possibility to output the generated model as XML, however, SemVis proceeds directly with the Java objects, that are generated by the Fresnel Engine¹.

The second step is performed by SemVis *Mapping Component*. With the additional presentation knowledge, defined in the *mapping-definition.n3* file, it adds presentation-

¹See also Section 4.7

paradigm-specific information. Yet the generated model is still not bound to a specific Visualization Platform. It exists in the shape of Java objects, what allows for easy computation on these objects.

The third and last transformation step is finally left to the *Java Emitter Templates* [JET] which do the rendering of the code for the final Visualization Platform (PSM) such as X3D or SVG. A template, which processes the Java objects from the second step and writes platform specific code can be created for arbitrary *Visualization Platforms*.

With the XML Jet template, it is also possible to again receive the generated model as XML output, which represents another PIM that can be transformed with other tools, for instance.

7.2. Applications of the MDA Pattern

The MDA pattern is applied multiple times by the SemVis architecture. This section describes the three transformation steps and compares the associated models: The RDF data, the Fresnel tree as a PIM, the SemVis Model as a further PIM and finally the Code for the Visualization Platform (here X3D² code) as the PSM. All transformation steps can not be done fully automatically, but require the input of additional knowledge to control the transformation (c.f. Figure 7.2).

As an example, data from a knowledge base about US presidents is taken and transformed step-by-step into X3D code. Please note, that uninteresting parts are left out and replaced by "...".

7.2.1. RDF Data (Domain Model)

The domain model of SemVis is represented by RDF data. The following excerpt of an RDF file (c.f. Example 7.1) defines an instance of the class `President`. The notation is RDF/XML.

²Extensible 3D — A standardized format for the description of 3D scenes [X3D]


```

1 ...
2
3 <rdf:Description
4   rdf:about='http://simile.../presidents/item#Abraham%20Lincoln'>
5   <a:label>Abraham Lincoln</a:label>
6   <b:type resource="#President" />
7   <d:imageURL>
8     http://upload.../Abraham_Lincoln.jpg
9   </d:imageURL>
10  <d:url>http://en.wikipedia.org/wiki/Abraham_Lincoln</d:url>
11  <d:presidency rdf:resource='16' />
12  <d:term>19</d:term>
13  <d:term>20</d:term>
14  <d:birth>1809-02-12</d:birth>
15  <d:death>1865-04-15</d:death>
16  <d:index>16</d:index>
17  <d:dieInOffice>yes</d:dieInOffice>
18  <d:party>Republican</d:party>
19  <d:birthPlace>Hardin County, Kentucky, USA</d:birthPlace>
20  <d:deathPlace>Washington, D.C.</d:deathPlace>
21  <d:religion>no affiliation</d:religion>
22  <d:birthLatLng>38.210509,-84.875859</d:birthLatLng>
23  <d:deathLatLng>38.895,-77.036667</d:deathLatLng>
24  <exhibit:origin>
25    http://simile.../presidents/presidents.html#Abraham%20Lincoln
26  </exhibit:origin>
27 </rdf:Description>
28
29 ... more Descriptions ...

```

Example 7.1. RDF Description of the Domain Model

7.2.2. First Transformation — RDF to Fresnel Tree

The first application of the MDA pattern is performed by the Fresnel Engine. The transformation is passed the RDF data (more precisely a selected fraction of it) and additionally the *fresnel-definition.n3* file that defines how to pick a certain view on the data and how the data has to be formatted. After the transformation, the structured data is then represented as a model of Java objects. Alternatively, the SIMILE Fresnel Engine allows the rendering of the Fresnel model as XML, conforming to the schema *fresnel-output.xsd*, that can be found in the appendix (c.f. Section A.1.1).

Example 7.3 shows an example for this XML version of the Fresnel *Platform Independent Model*, continuing the example from Section 7.2.1. Although only selected properties with exactly one value per property are picked by Example 7.2, the amount of necessary code has been drastically increased. The resource title is taken from the property `rdfs:label` (c.f. Example 7.2, line 11; Example 7.3 line 5). A link to the stylesheet that applies to all members of the group `mainGr` was added (c.f. Example 7.2, line 1-6; Example 7.3 line 4). Values of the property `imageUrl` are marked to be displayed as an image by the attribute `output-type="image"` (c.f. Example 7.2, line 24; Example 7.3 line 9). The `style-attributes` are turned into `class` attributes of the values (c.f. Example 7.2, line 26; Example 7.3 line 6). A new label, replacing the standard property label is added to the properties `religion` and `party` (c.f. Example 7.2, line 32; Example 7.3 line 27).

```
1 :mainGr rdf:type fresnel:Group ;
2 fresnel:stylesheetLink "styles/person.css" ;
3 fresnel:primaryClasses (
4   e:President
5 );
6 .
7
8 :personLens rdf:type fresnel:Lens ;
9 fresnel:purpose fresnel:defaultLens ;
10 fresnel:classLensDomain e:President;
11 fresnel:title rdfs:label;
12 fresnel:showProperties
13 (
14   d:imageURL
15   d:birth
16   d:party
17   d:religion
18 ) ;
19 fresnel:group :mainGr ;
20 .
21
22 :urlFormat rdf:type fresnel:Format ;
23 fresnel:propertyFormatDomain d:imageURL;
24 fresnel:value fresnel:image ;
25 fresnel:label fresnel:none ;
26 fresnel:propertyStyle "image" ;
27 fresnel:group :mainGr ;
28 .
29
30 :partyFormat rdf:type fresnel:Format ;
31 fresnel:propertyFormatDomain d:party ;
32 fresnel:label "Party: ^^xsd:string ;
33 fresnel:group :mainGr ;
34 .
```

Example 7.2. Example Fresnel Definition

```

1 <results>
2 <resource
3   uri="http://simile.../presidents/item#Abraham%20Lincoln">
4   <link>styles/person.css</link>
5   <title>Abraham Lincoln</title>
6   <property class="image"
7     uri="http://simile.../presidents/property#imageURL">
8     <values>
9       <value class="" output-type="image">
10        <title>
11          http://upload.wikimedia.../Abraham_Lincoln.jpg
12        </title>
13        </value>
14      </values>
15    </property>
16    <property uri="http://simile.../presidents/property#birth">
17      <values>
18        <value class="date">
19          <title>1809-02-12</title>
20        </value>
21      </values>
22    </property>
23    <property uri="http://simile.../presidents/property#party">
24      <label class="">
25        <content />
26        <title>Party:</title>
27      </label>
28      <values>
29        <value class="">
30          <title>Republican</title>
31        </value>
32      </values>
33    </property>
34    <property uri="http://simile.../presidents/property#religion">
35      <label class="">
36        <content />
37        <title>Religion:</title>
38      </label>
39      <values>
40        <value class="">
41          <title>no affiliation</title>
42        </value>
43      </values>
44    </property>
45    ... more properties ...
46  </resource>
47  ... more resources ...
48 </results>

```

Example 7.3. Fresnel Tree as XML Output

7.2.3. Second Transformation — Fresnel Tree to SemVis Model

The second application of the MDA pattern is performed by the *Mapping Component* of SemVis (c.f. Figure 7.2). The transformation is passed the Fresnel tree together with the *mapping-definition.n3* file that defines which Visual Variables are used to represent which facets of the data.

The SemVis transformation adds display-paradigm-specific information to the model from the last transformation step. The mapping definition files are evaluated and a new model that contains values for each Visual Variable is build. Although the values of the Visual Variables are now included in the model, the model is yet not limited to a specific Visualization Platform. If no value is provided by the mapping, the transformation process puts default values for all Visual Variables. In this way, the next transformation step (PIM to PSM) does not have to check for null values, but can expect values for all the variables. That way, the template code can be kept simple and the mapping logic can easily be handled on the SemVis level.

The option of transforming the XML Fresnel output tree with XSLT into the SemVis model has been rejected, since the complexity of the XSLT code soon became to high and only little means of structuring were possible. For instance, there was no object orientation supported. String conversions and time functions, as they were frequently needed, could have been provided with the help of extensions. But the parsing and evaluating of the mapping definitions, as well as the building of complex visualization structures, would have been very inconvenient and only possible with still more extensions to XSLT.

This led to the decision to work with the Fresnel Java objects instead of XSLT. Figure 7.3 shows the SemVis Java object model as a diagram of nested boxes. The wrapped Fresnel Java objects are shown on the right side of each visualization class. The model consists of `VisualizationElements` and of `VisualizationBoxes`, which implement container functions. For example, they are able to calculate their size recursively based on the size of their children³.

All `ResourceVisualizations` are `VisualizationElements` that stand for a primary class's instance and are controlled by the `VisualizationStructure`. The structure can modify the Visual Variable values of all its `ResourceVisualizations` and by this, it represents a coordinating instance. For a detailed description of the SemVis PIM classes, please refer to the JavaDoc.

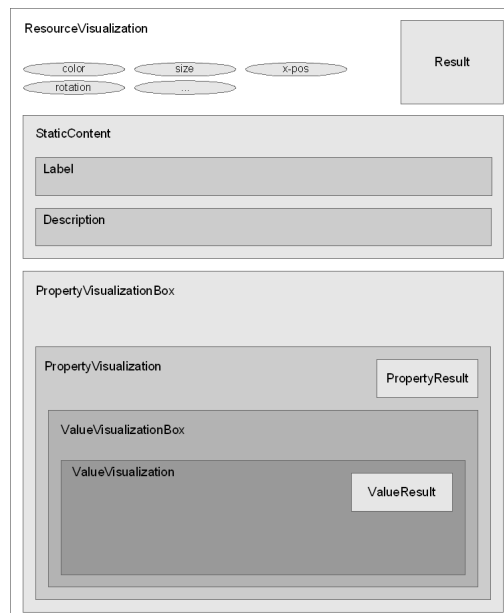


Figure 7.3. Illustration of the SemVis Model as Java Objects

³This behavior, which is something, that also was not possible with XSLT, is only rudimentary implemented, because here the integration of an existing advanced layout box model should be aimed at, similar to the [SWT] toolkit.

If it is necessary, SemVis can also output an XML version of the Java Object model that it generates, with the help of a template (c.f. Example 7.5 for an example of the XML code). It conforms to the schema *semvis.dtd* which can be found in Section A.1.2. Due to space limitations, the listing shows only an excerpt of the resulting code.

Example 7.4 defines an `ExplicitMapping` that maps the facet *party* to the Visual Variable `Color` and explicitly maps four discrete values. After the transformation, using the mapping, the `Color` value can be found on line 5 as an attribute of a `ResourceVisualization`.

```
1 default:partyMapping
2     a map:PropertyMapping ;
3     map:automatic "false"^^xsd:boolean ;
4     map:hasValueMappings default:Party2ColorValueMappings ;
5     map:sourceFacet d:party ;
6     map:targetVariable graphics:Color ;
7     map:targetValueNull graphics:grey ;
8 .
9
10 default:Party2ColorValueMappings
11     a map:ValueMappingContainer ;
12
13     rdf:_1 [ a map:ValueMapping ;
14             map:sourceValue "No Party" ;
15             map:targetValue graphics:white
16     ] ;
17     rdf:_2 [ a map:ValueMapping ;
18             map:sourceValue "Democratic-Republican" ;
19             map:targetValue graphics:lightRed
20     ] ;
21     rdf:_3 [ a map:ValueMapping ;
22             map:sourceValue "Democratic" ;
23             map:targetValue graphics:green
24     ] ;
25     rdf:_4 [ a map:ValueMapping ;
26             map:sourceValue "Republican" ;
27             map:targetValue graphics:red
28     ] ;
29
30 .
```

Example 7.4. Mapping of the Facet *party* to the Visual Variable `Color`

```

1 <VisualizationStructure>
2
3 <ResourceVisualization
4 <!-- Visual Variables-->
5   color="#FF00FF"
6   size="2"
7   xPos="0.56f"
8   yPos="1.5f"
9   zPos="2.0f"
10  rotation="0.5f"
11  transparency="0.5f" >
12
13  <StaticContent>
14    <Label content="Abraham Lincoln" />
15    <Description content="missing" />
16  </StaticContent>
17
18  <PropertyVisualizationBox>
19
20    <PropertyVisualization>
21      <Label content="Party:" />
22      <ValueVisualizationBox>
23        <ValueVisualization>
24          <Label content="Republican" />
25        </ValueVisualization>
26      </ValueVisualizationBox>
27    </PropertyVisualization>
28
29    ... more PropertyVisualizations ...
30
31  </PropertyVisualizationBox>
32
33 </ResourceVisualization>
34
35 ... more ResourceVisualizations ...
36
37 </VisualizationStructure>

```

Example 7.5. SemVis Model as XML Output

7.2.4. Third Transformation — SemVis Model to Code for Final Visualization Platform

One way to perform this last transformation step and generate the final presentation platforms code (PSM) was to have a render method in each class of the SemVis PIM. But this results in a subclass for each class of the PIM per presentation variant. Additionally we might even want to define multiple variants per presentation platform).

The currently chosen approach takes the model of Java objects from the last transformation step and hands it to a set of nested templates. This decouples the generation of the PIM from further transformations.

The transformation to the code for the presentation platforms is done with the help of the Java Emitter Templates [JET]. The JET template technology is a code writing mechanism and part of the *Eclipse Model To Text* [M2T] project. Alternatively, also any other template language could be employed.

The XML output of the SemVis model, already presented in the last section can be equally generated with JET. In this case it does not produce the final code, but another PIM. This XML output can then be taken as input to arbitrary XML transformation languages and by this an interface to, for instance, XSLT is provided.

```
1 ...
2 <Transform rotation="0 1 0 -0.6094943094478699">
3   <Transform translation="0.0 -63.503563 -35.0"
4     scale="1.3061224 1.3061224 1.3061224" >
5
6   <Billboard axisOfRotation="0 1 0">
7
8     <!-- Board behind resource -->
9     <Shape><Box size="12.0 24.72 0.05"/>
10    <Appearance>
11      <Material diffuseColor="1.0 0.0 0.0" />
12    </Appearance>
13  </Shape>
14
15  <LOD range="140">
16    <Transform scale="1 1 1" translation="0 0 0.2">
17
18      <!-- static content -->
19      <Transform translation="0.0 11.36 0">
20
21        <!-- label -->
22        <Transform translation="0.0 0.0 0">
23          <ProtoInstance name="text">
24            <fieldValue name="text" value="Abraham Lincoln"/>
25            <fieldValue name="color" value="1 1 1"/>
26            <fieldValue name="size" value="1.0" />
27            <fieldValue name="lodRange" value="130"/>
28          </ProtoInstance>
29        </Transform>
30      <!-- description -->
31    ...
32  </Transform></Transform><!-- end static content -->
33
34  <Transform translation="0 -1.0 0.2">
35  ...
```

Example 7.6. Generated X3D Code

```

1 ...<!-- property vis box (containing all properties) -->
2     <Shape>
3         <Box size="12.0 22.72 0.05"/>
4         <Appearance>
5             <Material diffuseColor="0.1 0.1 0.3" />
6         </Appearance>
7     </Shape>
8
9     <!-- single property (imageUrl) -->
10    <Transform scale="1 1 1" translation="0.0 3.5 0.2">
11        <Shape>
12            <Box size="12.0 15.719999 0.05"/>
13            <Appearance>
14                <Material diffuseColor="0.1 0.1 0.3" />
15            </Appearance>
16        </Shape>
17
18        <Transform scale="1 1 1" translation="0 0 0.2">
19
20            <!-- value vis box transformation -->
21            <Transform translation="0.0 0.0 0">
22
23                <Transform scale="1 1 1" translation="0 0 0.2">
24
25                    <Transform scale="12.0 15.719999 1"
26                        translation="0.0 0.0 0.0">
27                        <ProtoInstance name="image">
28                            <fieldValue name="imageUrl"
29                                value="http://upload.../Abraham_Lincoln.jpg" />
30                        </ProtoInstance>
31                    </Transform>
32
33                </Transform></Transform>
34            <!-- single property (birth) -->
35            ...
36                <!-- Single value -->
37                <Transform translation="0.0 0.0 0.0">
38                    <ProtoInstance name="textBoard">
39                        <fieldValue name="text" value="12.2.1809" />
40                        <fieldValue name="color" value="0.1 0.1 0.3" />
41                        <fieldValue name="size" value="12.0 1.0 1.0" />
42                    </ProtoInstance>
43                </Transform>
44            ...
45                <!-- label -->
46                <Transform translation="0.0 0.5 0">
47                    <ProtoInstance name="textBoard">
48                        <fieldValue name="text" value="Party: " />
49                        <fieldValue name="color" value="0.6 0.2 0.2" />
50                        <fieldValue name="size" value="12.0 1.0 0.05" />
51                    </ProtoInstance>
52                </Transform>
53            ...
54                <!-- Multiline text value -->
55                <ProtoInstance name="text">
56                    <fieldValue name="text" value="Republican"/>
57                    <fieldValue name="color" value="0.0 0.0 0.0"/>
58                    <fieldValue name="size" value="0.5" />
59                </ProtoInstance>
60            ...

```

Example 7.7. Generated X3D Code (continued)

Figure 7.4 shows a schematic picture of the X3D objects described by the code from the listing as it was rendered by a particular X3D Jet template. The static content is

drawn as a rectangle containing a title and a description. The dynamic content is attached vertically below and consists of a rectangle for each property. If the label is to be shown, it is displayed on top of the rectangle, followed by the values. Depending on the `fresnel:value` and the style attributes, different representations are chosen for the values. Image values are rendered as images and date values are rendered as text in a readable date format. Values of *ObjectProperties* are displayed recursively as resources themselves, and are arranged next to each other within the rectangle representing the property. Example 7.6 shows how the values for the Visual Variables have been integrated into the X3D code. A `Transform` node (c.f. Example 7.6, line 2) represents the resource (a president) and is attached the values for the spatial dimensions and the size (lines 2-4). The Visual Variable *Color* is assigned as a `diffuseColor` value of an `Appearance` node (line 11) that determines the appearance of the box framing the whole resource (line 9).

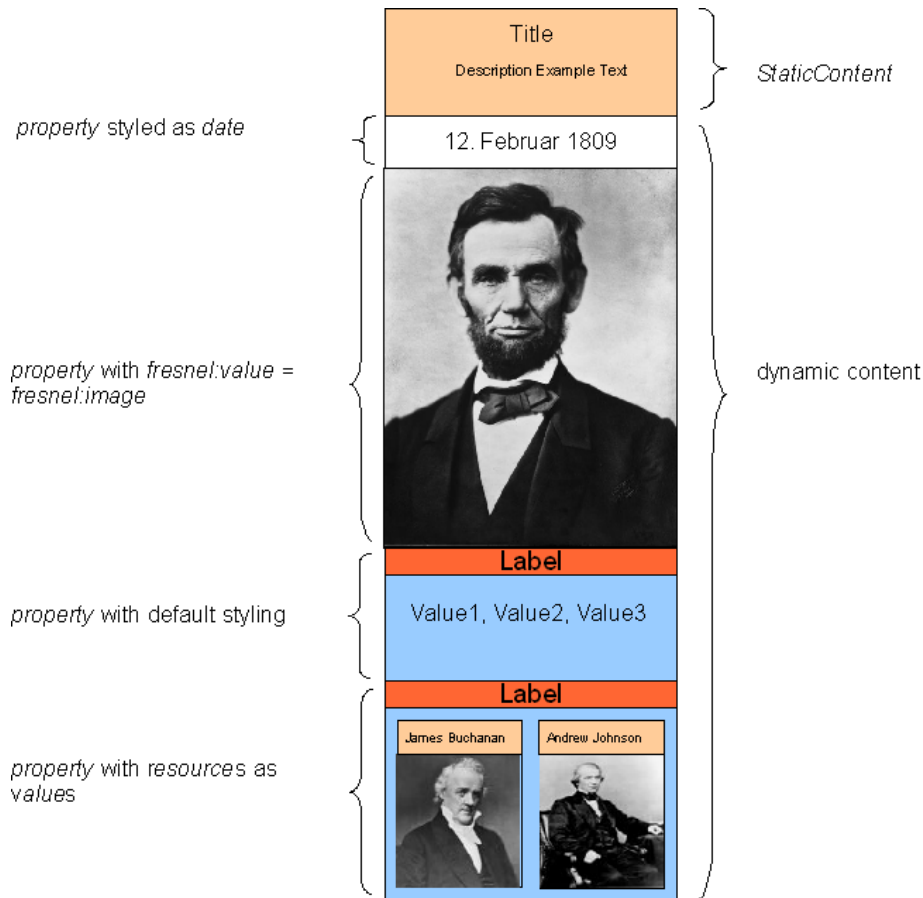


Figure 7.4. X3D Instantiation of the Abstract SemVis Box Model

7.3. Complete System Overview

Figure 7.5 shows, in addition to the Model-Driven Architecture aspects, the flow of data from and to all definition files and knowledge bases, as well as the two different GUI configuration components and Visualization Platforms. The sources of additional knowledge that are needed for the configuration components are described in detail in the next section (Section 7.4).

The core SemVis component can be configured via the *Admin Configuration GUI*, which can be used by the Admin to make SemVis load, generate and store display definitions. The Admin Configuration GUI supports the Admin with knowledge that the system gains from facet metrics, general graphical knowledge from the *Graphics.owl*

ontology and facts about the visualization platforms. The range of possibilities to which the User is limited (regarding further filtering and mapping) is defined in the *user-limitation.n3* file. An export function allows for the separate storage of the generated *fresnel-definition.n3* file to reuse it for other browsers, which also support Fresnel as their display vocabulary (c.f Section 4.6).

The *User Configuration GUI* is generated by SemVis and allows the User to proceed with customizing the visualization to his needs and pick several views on the data, within the limits defined by the Admin. The *User Configuration GUI* can either be embedded in the final Visualization Platform, if the platform supports interaction and dynamic reload of data, or build a frame around the static, more document like content and regenerate the content according to the Users settings, if necessary. The *User Configuration GUI* is able to load and save the Users settings and state of the browsing situation to a file.

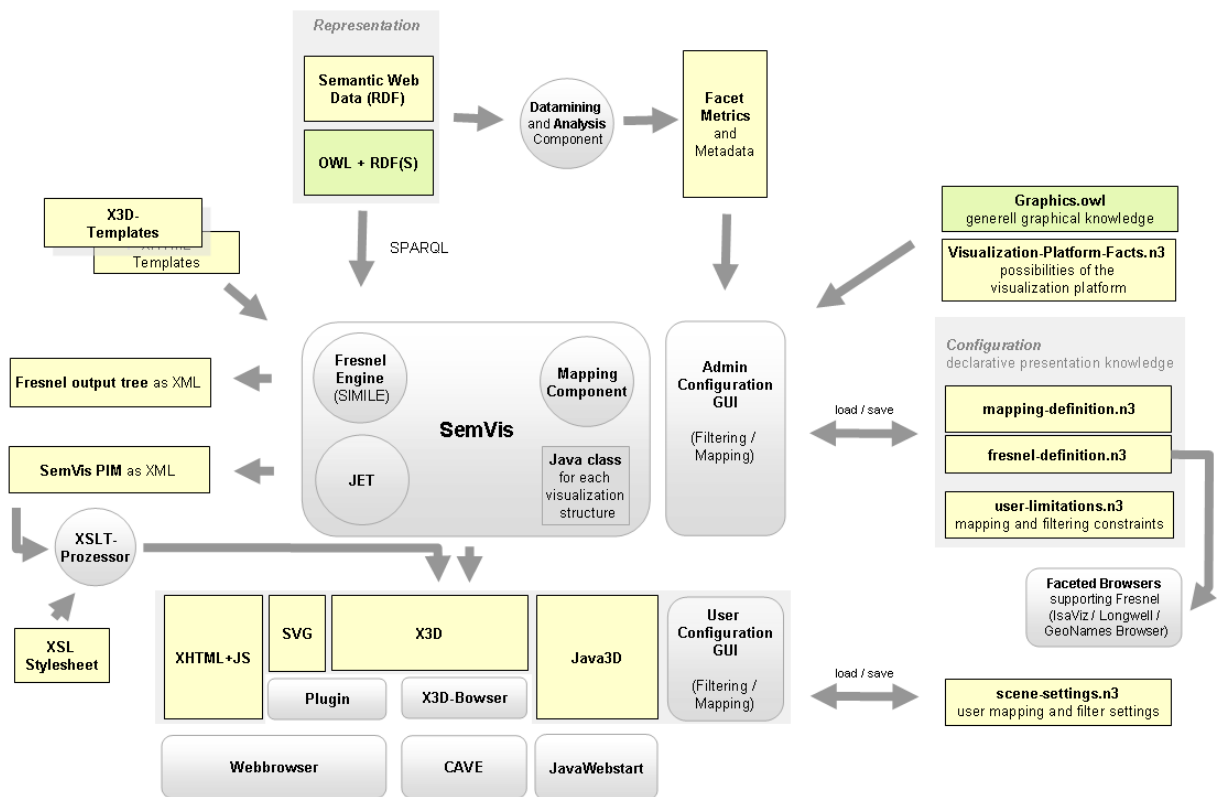


Figure 7.5. SemVis Complete Architecture

7.4. Additional Knowledge of the System

Besides the *fresnel-definition.n3* file, the *mapping-definition.n3* file and the templates, that are used as input to the transformations of the models, additional knowledge is required that serves as a knowledge base for SemVis to assist the Admin and the User in configuring these definition files. This comprises information on the capabilities of the Visualization Platforms, available Visualization Structures and their possibilities, general graphical knowledge and, meta information on the data that is to be visualized. Figure 7.6 gives an overview of the used definition files and their dependencies.

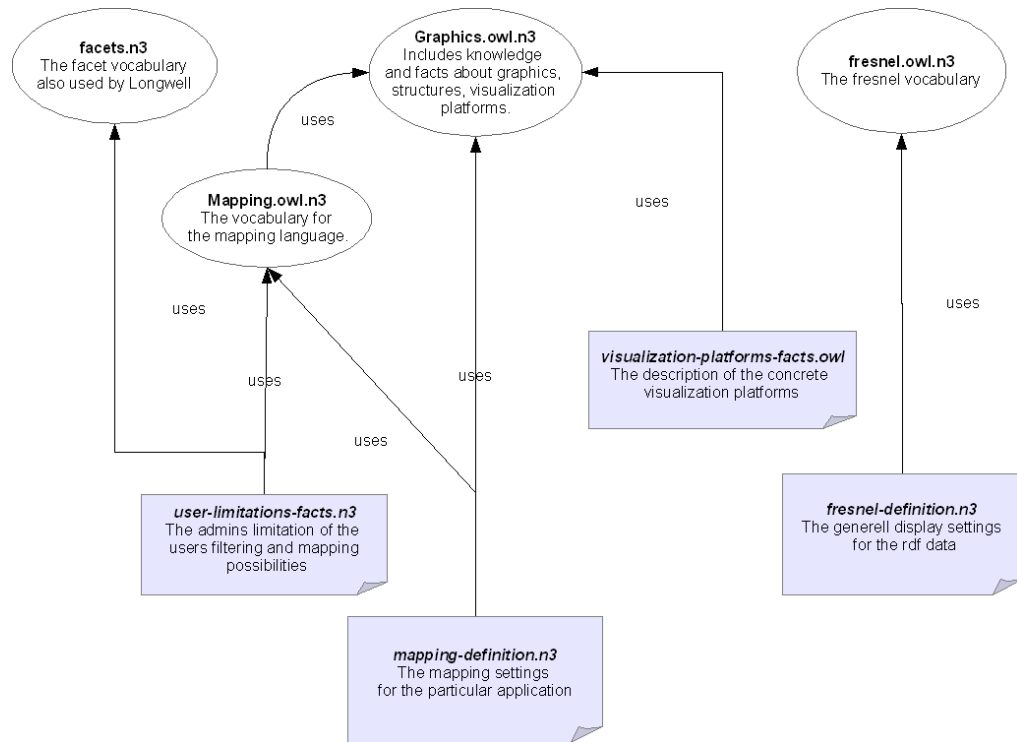


Figure 7.6. Overview of all Ontologies and Definition Files

7.4.1. Meta Data

Since we want to visualize semantic web data, there is no need for a special mechanism for meta information to describe the data. The data is already structured and self descriptive. If this is insufficient for the visualization purposes, the ontology can be extended and connected to upper level ontologies to gain more semantics. If we want to use other data, e.g. from databases, text-files or other semi structured sources, the data has to be converted to RDF for the moment. In the case of databases, at least information on simple data types is already available.

Meta data that has been retrieved from metrics and analysing the data is also made available to the system (c.f. Section 2.3.2).

7.4.2. Available Visualization Platforms and Presentation Scenarios

Each available platform must be described regarding to its possibilities. For example the graphical power needs to be described: How many spatial dimensions are there (2D or also 3D)? Is it possible to use color or not? Is animation and interaction possible? How high is the resolution? Where is the position of the user, i.e. is the situation immersive or not? Is sound also available? Equally the *Presentation Scenarios* have to be described. Scenarios could be the set up for a CAVE or a single screen of a home PC.

The definition of the available visualization platforms and scenarios with their abilities is noted in the file *visualization-platforms-facts.n3*. Also c.f. Example 7.8 and Example 7.9.

```

1 :XHTML
2     a     graphics:VisualizationPlatform ;
3     graphics:offersVisualVariable
4         graphics:Saturation , graphics:Height ,
5         graphics:Y-Pos , graphics:Roundness ,
6         graphics:Area , graphics:LineWidth ,
7         graphics:Texture , graphics:Transparency ,
8         graphics:Color , graphics:GreyValue ,
9         graphics:Order , graphics:Visibility ,
10        graphics:Contrast , graphics:Length ,
11        graphics:Brightness, graphics:Size ,
12        graphics:X-Pos;
13 :offersAnimation "false"^^xsd:boolean ;
14 :offersInteraction "false"^^xsd:boolean ;
15 :offersSpatialDimensions "2"^^xsd:integer ;
16 .
17
18 :X3D
19     a     graphics:VisualizationPlatform ;
20     graphics:offersVisualVariable
21         graphics:Saturation , graphics:Z-Pos ,
22         graphics:PositionInStructure , graphics:LineWidth ,
23         graphics:Orientation , graphics:Texture ,
24         graphics:Vibration , graphics:Color ,
25         graphics:Order , graphics:Blur ,
26         graphics:Contrast , graphics:Shape ,
27         graphics:Brightness , graphics:X-Pos ,
28         graphics:Volume , graphics:GlowingPulsation ,
29         graphics:BlinkingFrequency ,
30         graphics:Height , graphics:Roundness ,
31         graphics:Y-Pos , graphics:Area ,
32         graphics:RotationSpeed ,
33         graphics:Rotation , graphics:Transparency ,
34         graphics:GreyValue , graphics:Depth ,
35         graphics:DistanceFromViewer ,
36         graphics:TimeOfVisibility , graphics:Visibility ,
37         graphics:Length , graphics:Size;
38 :offersAnimation "true"^^xsd:boolean ;
39 :offersInteraction "true"^^xsd:boolean ;
40 :offersSpatialDimensions "3"^^xsd:integer ;
41 .

```

Example 7.8. Definition of XHTML and X3D as VisualizationPlatforms

```

1 :CAVE
2     a      graphics:PresentationScenario ;
3     graphics:isImmersive "true";
4     graphics:offersResolution "2400";
5     graphics:offersColors "16000000";
6     graphics:offersSpatialDimensions="3";
7 .
8
9 :SingleScreen
10    a      graphics:PresentationScenario ;
11    graphics:isImmersive "false";
12    graphics:offersResolution "1600";
13    graphics:offersColors "16000000";
14    graphics:offersSpatialDimensions="2";
15 .
16
17 :MobilePhone
18    a      graphics:PresentationScenario ;
19    graphics:isImmersive "false";
20    graphics:offersResolution "256";
21    graphics:offersColors "2";
22    graphics:offersSpatialDimensions="2";
23 .

```

Example 7.9. Definition of Presentation Scenarios

7.4.3. Available Visualization Structures

The system has to know which structures are there and which possibilities the structures offer. It can be stated, for instance, which spatial dimensions a structure offers and if it supports infinity in one or many directions. It can also be stated, if it is suitable for a CAVE, a *Single Screen* or a *Power Wall*⁴ installation.

```

1 :Helix
2     a      VisualizationStructure ;
3     graphics:offersVisualVariable
4         graphics:Height ,
5         graphics:Width ,
6         graphics:Depth ,
7         graphics:Color ;
8     graphics:infinitySupported "true"^^xsd:boolean;
9     graphics:suitableFor :CAVE ;
10 .

```

Example 7.10. Definition of a Visualization Structure

The vocabulary for the structures themselves can be found in the *Graphics.owl* ontology and the vocabulary for the mapping of facets to structures in the *Mapping.owl* ontology.

7.4.4. General Graphical Knowledge

General graphical facts do not have to be defined by the Programmer for each visualization. These facts can be used by SemVis to suggest suitable presetting for the configuration. Two examples are the ability of a *VisualVariable* to represent quantitative data or the dependency between two *VisualVariables*. The knowledge is contained in the *Graphics.owl* ontology that has been introduced in detail in Chapter 5.

⁴A projection on a (optionally bended) wall that offers some immersion and uses a 3D stereo projection

7.4.5. Limitations of the User

The User may only filter some facets and may only define mappings for a certain set of properties. This is necessary to confront the end user with a reasonable amount of choices. The restrictions are created by the Admin as described in Chapter 3.

In *Longwell*, the description of the filterable facets is stated with the property `facet:FacetSet` from the facet vocabulary⁵. I reuse this vocabulary for defining the filterable facets and store the facts in the file *user-limitations-facts.n3*. An example of this definition file can be found in Example 7.11. A `facet:FacetSet` needs to define a list of `rdf:Property`s to specify the facets which are displayed and their order (c.f. lines 4-5). With the property `facet:types`, a list of classes, to which this `facet:FacetSet` applies, can be declared. The value `facets:allTypes`, used here, makes this `facet:FacetSet` apply to any type.

```

1 :semVisExampleFacets rdf:type facet:FacetSet ;
2   facet:types facet:allTypes ;
3   facet:facets (
4     hist:hasStartTime
5     mid:relatedToFieldOf
6   ) ;
7 .

```

Example 7.11. User Limitations of the Filtering

Since the property `facet:facets` points to a list, the order of the entry can be used as the order of the facets in the GUI.

The limitations the Admin makes to the set of mappable properties are also stored in the file *user-limitations-facts.n3*, and are defined in analogy to the limitation of filterable facets. For this purpose, the mapping vocabulary contains the necessary terms `map:MapablePropertiesSet` and `map:mapableProperties`. An example of a `map:MapablePropertiesSet` instance is given in Example 7.12. The only property `map:mapableProperties` points to a list of `rdf:Property`s. The mapping settings are specified application-wide.

```

1 :semVisExampleMapableProperties rdf:type map:MapablePropertiesSet ;
2   map:mapableProperties (
3     hist:hasStartTime
4     mid:relatedToFieldOf
5     mid:importance
6   ) ;
7 .

```

Example 7.12. User Limitations of the Mapping

7.4.6. Initial Settings

The Admin provides initial settings for the User regarding the *Filtering*, the *Structuring*, the selection of Primaries and the display settings, i.e. the mapping. These are defined as SPARQL queries for the filtering settings and with the Fresnel vocabulary for Primaries, *Structuring* and *Formatting*. For the initial mapping settings, the SemVis mapping vocabulary is used.

⁵Namespace: <http://simile.mit.edu/2006/01/ontologies/fresnel-facets>

7.5. Comparison to the Graphical Modelling Framework — GMF

The architecture of SemVis has some similarities to the architecture of the Graphical Modeling Framework for Eclipse [GMF]. Especially for those, who are familiar with the GMF, this section offers a comparison of the working principles of SemVis and the GMF. For those who do not know the framework, the next paragraph gives a short introduction.

7.5.1. Short Overview of the Graphical Modelling Framework

The GMF enables programmers to quickly build a graphical editor based on a model of the domain and an additional description of tools and graphics of the editor.

At the beginning stands the *Domain Model*, which is represented in *Ecore*, the meta model that is used by the *Eclipse Modelling Framework* (EMF). Additionally, there might be existing descriptions of graphical elements and tools. If not, the GMF can help with creating these definition files by deriving default settings for tools and graphics based on the *Domain Model* (c.f. Figure 7.7). GMF distinguishes between figures, which can be any graphical element, such as *Line* or *Rectangle*, and *Nodes*, which carry semantics such as *Relation* or *InfoBox*.

After further refining the derived *Graphical Definition* and *Tool Definition*, the domain elements can be mapped to the tools and graphics in the *Mapping Model*.

If this is done, the GMF can use the *Mapping Model* to automatically create a *Generator Model* (the PIM, describing the diagram editor) and proceed with generating the editor code (the PSM).

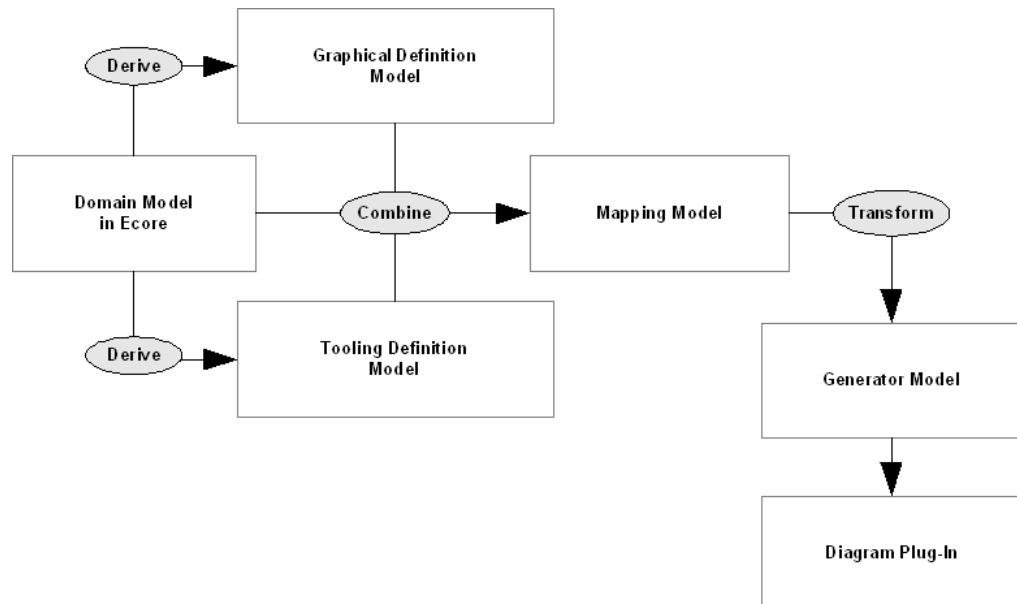


Figure 7.7. Architecture of the Graphical Modelling Framework

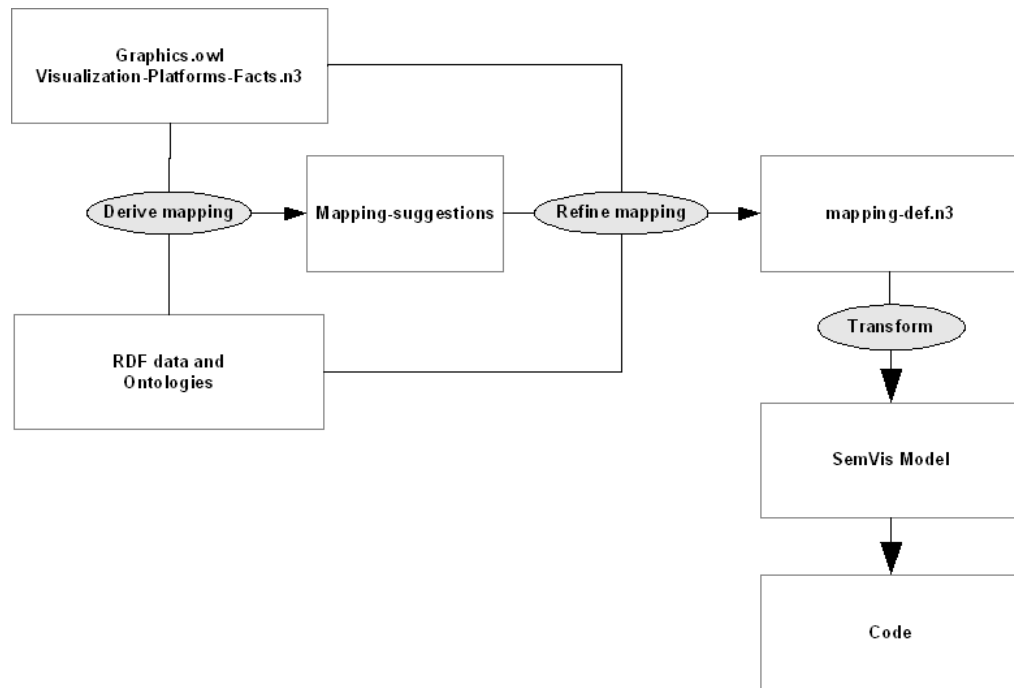


Figure 7.8. Architecture of SemVis

7.5.2. Comparison to SemVis

Figure 7.8 shows the SemVis architecture from an GMF point of view that allows to compare both frameworks to each other. The filtering aspects of SemVis are left out for a simplification.

The *Domain Model* of the GMF has its equivalent in the domain ontologies of SemVis. While the GMF expects the model to be written in Ecore, SemVis expects it to be written in RDF(S) and OWL. The *Graphics.owl* ontology and further display knowledge files play the role of the *Graphical Definition Model*. Both frameworks aim at creating an interactive GUI, but since SemVis does not generate editors but browsers and documents, it needs no *Tooling Definition*.

The *Derive* tasks of the GMF framework, which use the *Domain Model* to automatically suggest initial *Tooling Definition* and *Graphical Definition*, have a similar function as the components of SemVis, which suggest settings for the mapping and filtering configuration. However, in SemVis the system does not derive the graphical knowledge from the domain model, but uses both to derive suggestions for the mapping. The suggestions are then refined by the Admin into the final *mapping-definition.n3*.

The *Mapping Model* of the GMF has its counterpart in the *mapping-definition.n3* file. The resulting PIM of the mapping that is called *Generator Model* in GMF is represented by the *SemVis Model* (The transformation steps are simplified in the diagram). Both frameworks require a new build of the GUI after changes have been made to the mapping. In the case of SemVis it would be beneficial to allow a dynamic change of the mapping at runtime, continuously changing the view.

It is also common to both frameworks, that they can generate code for multiple end presentation platforms. The GMFs *Generator Model* can be used to generate code in other languages than Java and the templates of SemVis also allow to output code in arbitrary languages. But in the question of separation of content and presentation knowledge, SemVis and the GMF work differently. While the GMF separates the data from the

presentation code and loads the data into the editor, SemVis combines the presentation code and the data for the time being.

Chapter 8. Visualization Platforms

SemVis is independent of a specific Visualization Platform and can be used to generate code for several platforms by using a template mechanism. Possible output formats include XHTML, SVG, X3D and plain text. This chapter shows examples of these presentation platforms and discusses their advantages and disadvantages in transporting information.

8.1. Extensible 3D (X3D)

X3D is the successor of VRML, the *Virtual Reality Markup Language*, already introduced in 1995. It became an ISO standard for the description of 3D scenes in 2004, [X3D]. The internal representation of the model as Java Objects, allows for complex computations of the structure and mutual influences between the model elements, which is particularly important for the positioning of elements in 3D space.

Since most final output devices are two dimensional, a 3D visualization does only in some cases offer an added value. Often a 2D representation is better readable on a 2D display device. Yet the third dimension is a valuable additional Visual Variable that can be assigned semantics to. A 3D visualization of RDF data can be used in a virtual reality environment like a CAVE¹, where these advantages can be directly used. Here the impression of a virtual world constructed from knowledge could be generated.

However, the advantages, gained from the third dimension are easily absorbed if the navigation is not appropriate and does not support the user in doing her tasks actively, since navigation in three dimensional scenes is disproportionately more complicated. Although some real 3D input devices exist, users have to learn using them before they can start exploring or manipulating the scene as easy as in 2D. Another drawback is the still bad readability of text in 3D scenes. This is partly due to insufficient resolution and antialiasing capabilities, but also an inherent problem of the perspective squeezing of the text. Billboard² behaviors can help to overcome these problems in some cases.

Since X3D is a documentation format, it describes the scene in a declarative way. Still complex interaction and dynamic creation of scene elements can be achieved via the *Scene Access Interface (SAI)*. *ECMAScript*³ or Java classes can be addressed with this interface.

Figure 8.1 shows a screenshot of a generated X3D scene, corresponding to the examples from Section 7.2. The other, Figure 8.2 displays data from the history domain. Both scenes are viewed with the *Flux Player*⁴.

¹CAVE — Cave Automatic Virtual Environment. A cubic room with projectors directed to most of its sides to enable an immersive virtual reality.

²Billboard is the behavior of constantly orientation to the user. This way, the objects become viewpoint independent (in regard to the rotation).

³The standardized variant of JavaScript

⁴*Flux Player* is a free player for X3D: <http://www.mediamachines.com>

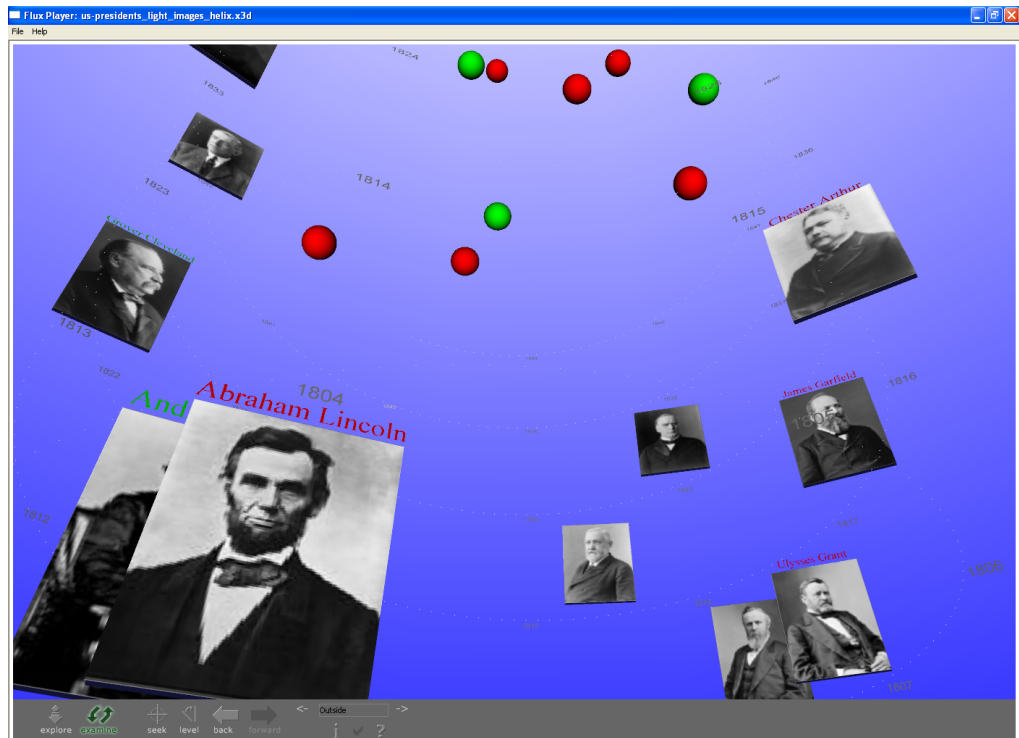


Figure 8.1. Example of X3D Output — US Presidents

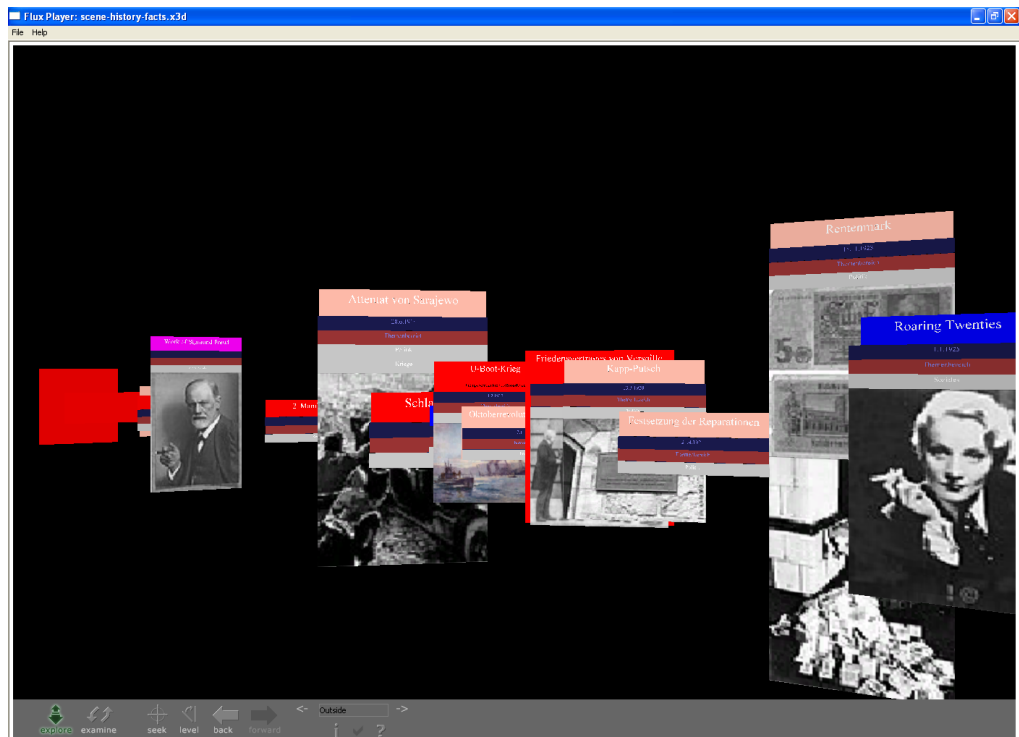


Figure 8.2. Example of X3D Output — Events of History

8.2. Scalable Vector Graphics (SVG)

Scalable Vector Graphics is a standard for the description of vector graphics in 2D, since 2001 defined by an W3C recommendation, [SVG]. It uses an XML syntax and

integrates [SMIL] to define animations. Interaction can be achieved with the help of ECMA Scripts³ that can access the DOM of the vector graphic. The definition of styling attributes with CSS is supported.

Figure 8.3 shows the example data (US presidents) rendered from SVG. The color states the *party*, the president belongs to and the y-position represents his affiliation to a particular *religion*. The facets *term*, *party*, *religion*, *birth*, *birthPlace*, *death* and *deathPlace* are additionally rendered as text. Note that vT values from the facet *term* are styled differently than the others.

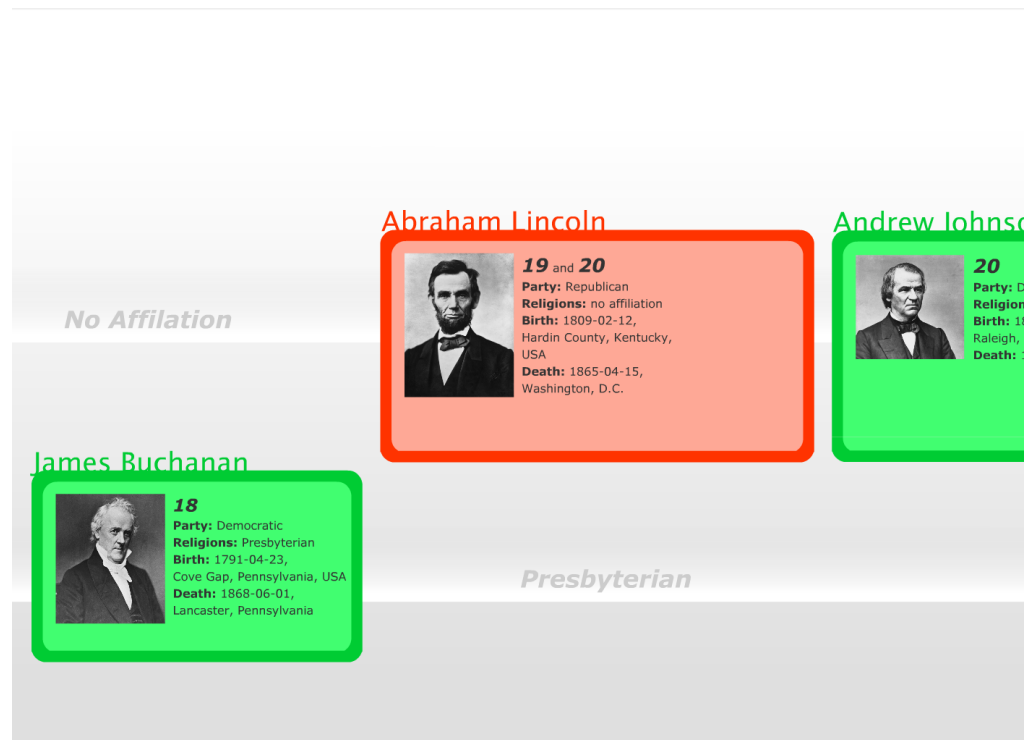


Figure 8.3. Example of SVG Output — US Presidents

8.3. XHTML + CSS

The output of XHTML and CSS files can be generated with an XHTML JET template. Again ECMA Script³ can be integrated to implement interactive features and animation.

8.4. Text

The size and color of simple text can be used to carry meaning in addition to the texts content. This is implemented by TagClouds⁵, for example. SemVis can also be used to easily generate such a formatted text, using an XSL-FO⁶ template for JET. An example, again showing data from the domain of history, is presented in Figure 8.4. The *font-color* and the *font-size* are both mapped to the *importance* of historical events⁷.

⁵Collections of tags written in different size depending on their number of usages

⁶XSL-FO — XSL Formatting Objects: A markup language for formatting of documents in XML

⁷The importance values are assigned randomly to the example data and do not reflect real facts.

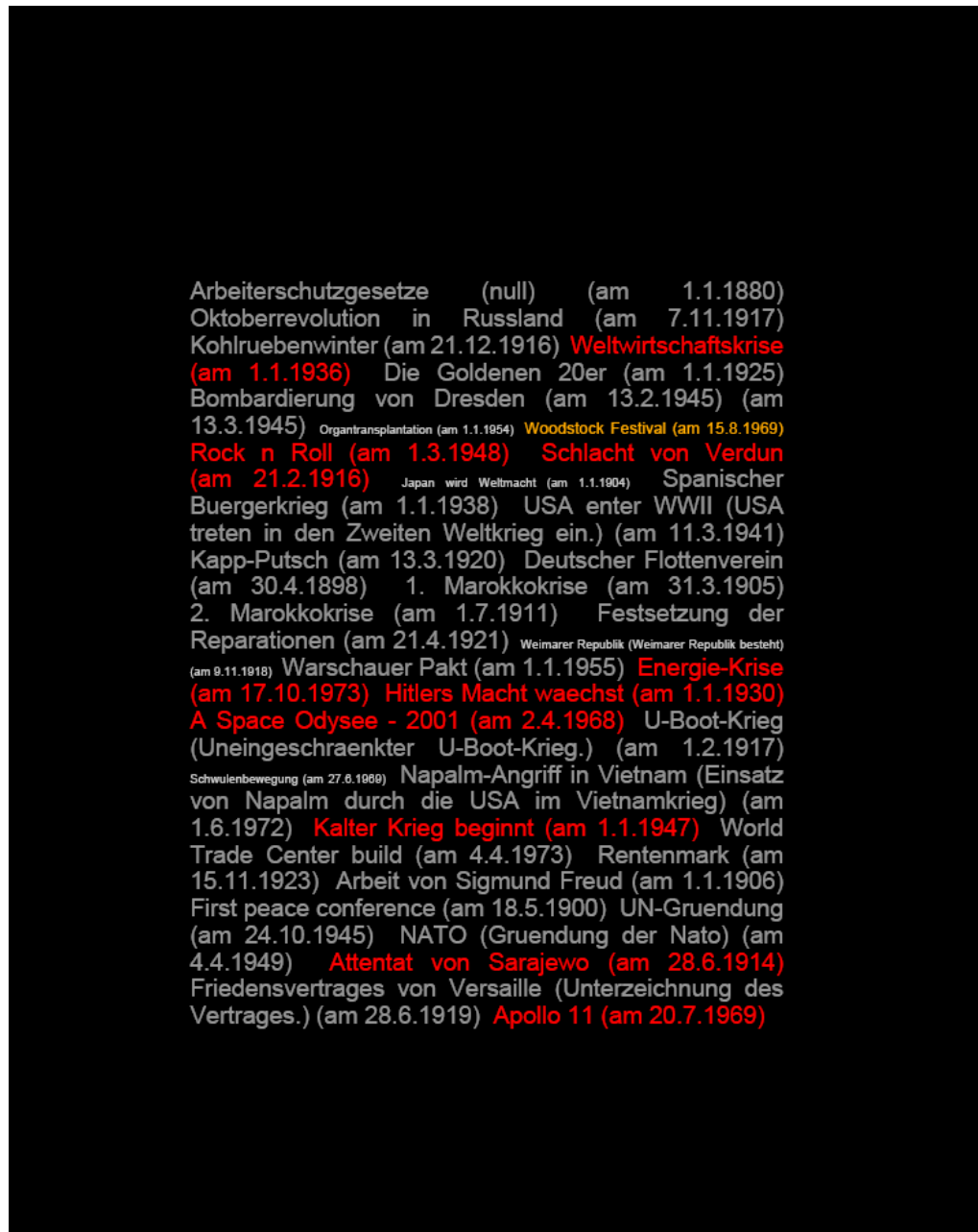


Figure 8.4. Example of PDF Output — Events of History

Chapter 9. Outlook and Conclusion

This section mentions ideas, not implemented nor described in detail, defines what SemVis is not intended to do and concludes which of the initial goals could be fulfilled by SemVis.

9.1. Advanced Mapping Vocabulary

The mapping vocabulary has to be expanded and refined in order to allow the handling of multiple source facet value types at the same time (e.g. the facet *happendIn*, pointing to *years*, *decades* etc.). If possible OWL DL solutions should be preferred to the OWL FULL restrictions that are utilized. Furthermore MathML as a standard vocabulary for the mathematical aspects of visualization should be reused.

9.2. Reusing Standardized Ontologies

As soon as mature, universal and generally accepted ontologies are established in the field of visualization or graphics, these should replace the basic *Graphics.owl* ontology that was constructed for this work in absence of a usable standard ontology. However, the ideas of SemVis may be taken as a recommendation to build upon or may be integrated in such a future generic visualization ontology.

9.3. Enabling Dynamic, Interaction and Animation

Little efforts have been spent on the support of interaction and dynamic features of the final presentation at the present state. However, this is important for a convenient usage of SemVis as a browser or editor.

Table 9.1 summarizes the constraints of the presentation platforms in regard to dynamic criteria. All of the three platforms mentioned below do generally support animation, interaction and even dynamic content loading.

	X3D	Java3D	SVG
Animation while viewing the presentation	Yes, via event routes and/or ECMAScript or Java classes using the SAI	Yes, with Behaviors	Yes, with SMIL
Interaction while viewing the presentation	Yes, via event routes and/or ECMAScript or Java classes using the SAI	Yes, with Behaviors	Yes, via the DOM using ECMAScript
Loading of additional data, creating nodes in the scene	Yes, via ECMAScript or Java classes using the SAI	Yes, directly via the Java3D API. Some capabilities have to be set	Yes, via the DOM using ECMAScript

Table 9.1. Comparison of Dynamic Possibilities between X3D, Java3D and SVG

Especially the filtering of contents by restriction of facet values should be also possible via the user interface and lead to a dynamic change of the presented data while changing

the filter settings or mapping. The view of the user should be continuously changed within an animation, to preserve the users orientation.

To achieve this, the staged transformation pipeline has to be modified in a way that small parts can be separately exchanged and the transformation is incrementally performed. Also the use of a caching mechanism would be advantageous to avoid expensively calculating and querying unmodified parts of the visualization again and again.

An important way of interaction is the editing of the presented data. A prerequisite for the editor functionality is the existence of an inverse transformation to transform the changes in the visualization back to the domain model.

9.4. Implementation and Evaluation

The prototypical implementation needs to be updated and the GUI as an essential part of the framework needs to be enforced. Although the mapping and the Fresnel definition can be done with a text editor, or more conveniently (but only partially) with the universal RDF(S)/OWL editor Protégé, a tailored GUI is required to offer the full benefit of SemVis. This is the most urgent step to take in order to evaluate the usefulness for domain experts.

9.5. Conclusion

After having shown the general need for a generic solution for visualization of structured data, we presented SemVis as a flexible framework for the definition of presentation knowledge for visualization on several visualization platforms. As this is work in progress, the vocabularies have to be refined and solutions to support also interactive features have to be found. Still the main goals could be fulfilled:

The visualization is highly flexible and configurable due to an exchangeable display definition and a variable visualization platform.

Two vocabularies have been developed for this purpose. First, the *Mapping.owl* ontology for a declarative, presentation-platform-independent definition of mappings from facets of the data to Visual Variables and second, the *Graphics.owl* ontology, providing general terms of visualization and graphics. The use of ontologies for this purpose increases reusability, as does the integration of current standards from the semantic web, such as the Fresnel RDF display vocabulary and the query language SPARQL.

The visualization can be applied to arbitrary structured data in the RDF format, at the cost of writing new mapping and configuration files, but making changes to the source code unnecessary. This way domain independence is achieved.

Although the visualization process is semi-automatic, it is to a high degree supported by the system and allows for fit-tailored visualizations. The GUI, supporting the user, can make use of common visualization knowledge (such as facts from cognition) and characteristics of the data (such as metrics and structure) to reduce the set of possible mapping choices for the user.

This way the drawbacks of the former framework have been resolved and additionally platform independence has been gained.

Although only implemented in pieces at the time being, the basis for a framework has been established that can be used to give format free, structured data, coming from the semantic web or other yet unknown sources, a shape that is human understandable while still preserving the advantages of complex filtering that is possible with machine understandable data.

Glossary

Facet	A property of a class in its role of slicing the available instances of this class
Facet Structure	The topology formed by a relation between the facet values. There can be several relations.
Filter	A restriction on property values, that reduces the initial set of instances
Fresnel	An RDF display vocabulary [FRESNEL]
Mapping	If not further described, a mapping means the mapping from a facet to Visual Variable
Primary	An ontology class, that is intended to be used as a starting point and has increased importance for the presentation (c.f. the section called “Starting Points (Primaries)”)
Template	Fraction of code, which is combined with dynamic values by a template engine
Value Range	The span from the minimum to the maximum value of a facet or a Visual Variable
Visualization Platform	The final presentation language for which SemVis outputs the platform specific code
Visualization Structure	A visualization structure describes the characteristics of the structure that is the result of a technique of visualization, including the shape, and suitability for different presentation scenarios, but not the technique itself.
Visual Variable	A dimension or property of a graphical element according to [Ber81]. Also referred to as elements of a graphical vocabulary, according to [Zeh04]

Files

fresnel-definition.n3	An example file for the display settings of Fresnel.
Graphics.owl	General graphical knowledge and facts including <code>VisualVariables</code> and their <code>VisualValues</code> . URL: " http://www.polowinski.de/ontologies/Graphics.owl "
Mapping.owl	The SemVis mapping vocabulary
mapping-definition.n3	The facts about a specific mapping. Uses the vocabulary from <i>Mapping.owl</i>
user-limitations-facts.n3	The limitations the Admin makes to the user regarding properties for filtering and mappable properties
visualization-platforms-facts.n3	A description of the graphical power of a platform. URL: " http://www.polowinski.de/ontologies/Visualization-Platforms.owl "

Bibliography

- [ARS06] 2006. Avis, Rana, Shu. *Investigating Visualization Ontologies*. Proceedings of the UK e-Science All Hands Conference 2006. National e-Science Centre. 249 - 257. <http://www.allhands.org.uk/2006/>. 0-9553988-0-0.
- [Ber81] 1981. J. Bertin. *Graphics and Graphic Information Processing*. Walter de Gruyter, Berlin.
- [BLP05] 2005. Bizer, Lee, Pietriga. *Fresnel — a browser-independent presentation vocabulary for rdf*. End user semantic web interaction workshop at the ISWC 2005, Galway, Ireland. <http://simile.mit.edu/repository/fresnel/trunk/docs/papers/wsISWC/fresnel.pdf>.
- [DW06] 2006. Dachsel, Weiland. *TimeZoom: A Flexible Detail and Context Timeline*. ACM Press, Montréal (Québec, Canada). 1-59593-298-4.
- [Del.icio.us] *Del.icio.us*. <http://del.icio.us>.
- [DBD04] 2004. Duke, D.J.; Brodlie, K.W.; Duce, D.A.. *Building an Ontology of Visualization*. Visualization, 2004. IEEE, Volume , Issue , 10-15 Oct. 2004. 7.
- [CM03] 2003. T. Catarci and T. Di Mascio and E. Franconi and G. Santucci and S. Tessaris. *An ontology based visual tool for query formulation support*. citeseer.ist.psu.edu/catarci04ontology.html.
- [Chi00] 2000. Ed H. Chi. *A Taxonomy of Visualization Techniques Using the Data State Reference Model*. Proceedings of the IEEE Symposium on Information Visualization 2000. 69. 0-7695-0804-9.
- [Ebay Express] . *Ebay express*. <http://search.express.ebay.com>.
- [FH06] 2006. Fadhil, Haarslev. *GLOO: A Graphical Query Language for OWL Ontologies*. Proceedings of the 2006 International Workshop on OWL: Experiences and Directions 2006 (OWLED-2006), Athens, Georgia, USA, Nov. 10-11, 2006..
- [FLAMENCO] . *FLAMENCO*. <http://flamenco.berkeley.edu/>.
- [FlickrR] *Flickr*. <http://www.flickr.com/>.
- [FRESNEL] 2005. *FRESNEL*. <http://www.w3.org/2005/04/fresnel-info/>.
- [FSL] 2005. *Fresnel Selector Language for RDF (FSL)*. <http://www.w3.org/2005/04/fresnel-info/fsl/>.
- [GH05] 2005. Hannes Gassert and Andreas Harth. *From Graph to GUI: Displaying RDF Data from the Web with Arago*. citeseer.ist.psu.edu/731287.html.
- [Geonames Browser] Emmanuel Pietriga. *Geonames Browser*. <http://zvtm.svn.sourceforge.net/viewvc/zvtm/net/claribole/gnb/>.
- [GMF] *GMF*. <http://www.eclipse.org/gmf/>.
- [Gnowsis] *Gnowsis*. <http://www.gnowsis.org/>.
- [Google Base] . *Google base BETA*. <http://base.google.com/>.
- [HMS00] 2000. *Graph Visualization and Navigation in Information Visualization: a Survey*. Ivan Herman, Guy Melan and M. Scott Marshall. IEEE Transactions on Visualization and Computer Graphics. citeseer.ist.psu.edu/herman00graph.html.
- [IsaViz] *IsaViz: A Visual Authoring Tool for RDF*. <http://www.w3.org/2001/11/IsaViz/>.

- [Java3D] *Java3D*. <http://www.j3d.org/>.
- [Jena] *Jena*. <http://jena.sourceforge.net/>.
- [JET] *JET*. <http://www.eclipse.org/modeling/m2t/?project=jet#jet>.
- [JFresnel] *JFresnel*. <http://jfresnel.gforge.inria.fr/>.
- [JS91] 1991. Johnson, B. and Shneiderman, B.. *TreeMaps: A space-filling approach to the visualization of hierarchical information structures*. Proc. Visualization '91, pp. 284-291.
- [BHL01] May 17, 2001. Tim Berners-Lee, James Hendler and Ora Lassila. http://www.sciam.com/print_version.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21.
- [LONGWELL] *Longwell*. <http://simile.mit.edu/wiki/Longwell>.
- [Mac86] 1986. Mackinlay. *Automating the Design of Graphical Presentations*. In ACM Transactions on Graphics, Vol. 5, No. 2, S.110-141.
- [Mambo] 2007. Dachsel, Rainmund. *Mambo — Mobile Aspect-based Media BrOwser*. <http://www-mmt.inf.tu-dresden.de/Forschung/Projekte/INPERIC/Results/Mambo/index.xhtml>.
- [MediaWiki] *MediaWiki*. <http://www.mediawiki.org/wiki/MediaWiki>.
- [M2T] *Model to Text (M2T)*. <http://www.eclipse.org/modeling/m2t>.
- [MathML] *MathML*. <http://www.w3.org/Math/>.
- [M-SPACE] 2004. *M-Space*. <http://beta.mspace.fm/>.
- [N3] 2006. Tim Berners-Lee. *Notation3 (N3) Syntax — A readable language for data on the Web*. <http://www.w3.org/DesignIssues/Notation3.html>.
- [N-Ary] 2006. *Defining N-ary Relations on the semantic web*. <http://www.w3.org/TR/swbp-n-aryRelations/>. Natasha Noy, Alan Rector.
- [MDA] *OMG*. <http://www.omg.org/mda/>.
- [ODD06] 2006. E. Oren and R. Delbru and S. Decker. *Extending faceted navigation for RDF data*. ISWC. citeseer.ist.psu.edu/oren06extending.html.
- [OWL] 2004. *OWL Web Ontology Language Reference*. <http://www.w3.org/TR/owl-ref/>. <http://www.w3.org/TR/owl-features/>.
- [Pol06] 2006. Polowinski, Jan. *Visualisierung großer Datenmengen im 3D-Raum*. Belgarbeit an der TU Dresden.
- [Prd91] 1991. *Software*. Prieto-Díaz, Rubén. <http://doi.acm.org/10.1145/103167.103176>.
- [Pri00] 2000. Priss, Uta. *Faceted Information Representation*. <http://citeseer.ist.psu.edu/priss00faceted.html>.
- [PROTÉGÉ] *Protégé*. <http://protege.stanford.edu/>.
- [QHK03] D. Quan and D. Huynh and D. Karger. 2003. *Haystack: A Platform for Authoring End User semantic web Applications*. citeseer.ist.psu.edu/quant03haystack.html.
- [RDF] *Resource Description Framework*. <http://www.w3.org/RDF/>.
- [RDFS] *RDF Schema*. <http://www.w3.org/TR/rdf-schema>.
- [Relation Browser] Moritz Stefaner. *Relation Browser*. <http://der-mo.net/relation-browser/>.

- [Rut05] L. Rutledge and J. van Ossenbruggen and L. Hardman. *Making RDF presentable: integrated global and local semantic Web browsing*. 2005. citeseer.ist.psu.edu/rutledge05making.html.
- [Sam07] 2007. *Branchenreport: Semantische Technologien in den Life Sciences*. Matthias Samwald. Reihe Branchenreport der Semantic Web School, Ausgabe 20.
- [SK06] Mc Schraefel, David Karger. *The pathetic fallacy of RDF*. 2006. Position Paper for SWUI06.
- [Semantic MediaWiki] *Semantic MediaWiki*. http://ontoworld.org/wiki/Semantic_MediaWiki.
- [SESAME] *openrdf.org*. <http://www.openrdf.org/>.
- [SIMILE] <http://simile.mit.edu/>. 2005. *The Simile Project*.
- [Sin05] 2005. Rashmi Sinha. *Weblog entry, detailed analysis of Google Base..* http://www.rashmisinha.com/archives/05_11/.
- [SMIL] *SMIL*. <http://www.w3.org/AudioVideo/>.
- [SCM+06] 2006. Greg Smith, Mary Czerwinski, Brian Meyers, Daniel Robbins, George Robertson, Desney S. Tan. *FacetMap: A Scalable Search and Browse Visualization*. IEEE Transactions on visualization and computer graphics, vol.12 , No. 5, september/october 2006.
- [SPARQL] *SPARQL*. <http://www.w3.org/TR/rdf-sparql-query/>.
- [SVG] 2004. *SVG — W3C Recommendation 14 January 2003*. <http://www.w3.org/TR/SVG/>.
- [SWT] *SWT: The Standard Widget Toolkit*. <http://www.eclipse.org/swt/>.
- [Wal03] Norman Walsh. *RDF Twig: Accessing RDF Graphs in XSLT*. 7. Aug 2003. Sun Microsystems, inc..
- [Woo96] 1996. *The null object pattern*. B. Woolf. In Group 5: Design Patterns, PLoP '96, Robert Allerton Park and Conference Center, University of Illinois at Urbana-Champaign, Monticello, Illinois, Sept. 36 1996. PLoP '96.. citeseer.ist.psu.edu/woolf96null.html.
- [X3D] 2004. *X3D — ISO/IEC 19775-1:2004*. http://www.web3d.org/x3d/specifications/x3d_specification.html.
- [Zeh04] 2004. Zehler, Thomas. *Software-Visualisierung*. (BTU) Report: ViSEK/045/D Version 1.30 04.02.2004.

Appendix A.

This chapter lists the schemata, classes and example files in detail.

A.1. Schemata

This section shows the schemata, used within SemVis.

A.1.1. Fresnel Tree Output Schema

The following schema is used by the Fresnel Engine.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:annotation>
    <xsd:documentation xml:lang="en">
      Schema for Fresnel output results.
    </xsd:documentation>
  </xsd:annotation>

  <xsd:element name="results" type="FresnelResults"/>

  <xsd:complexType name="FresnelResults">
    <xsd:sequence>
      <xsd:element name="resource" type="FresnelResource"
minOccurs="0"
      maxOccurs="unbounded"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="FresnelResource">
    <xsd:sequence>
      <xsd:element name="content" type="FresnelContents" minOccurs="0"
      maxOccurs="1"/>
      <xsd:element name="title" type="xsd:string" minOccurs="1"
      maxOccurs="1"/>
      <xsd:element name="property" type="FresnelProperty"
minOccurs="1"
      maxOccurs="unbounded"/>
    </xsd:sequence>
    <xsd:attribute name="class" type="xsd:string"/>
    <xsd:attribute name="uri" type="xsd:string"/>
  </xsd:complexType>

  <xsd:complexType name="FresnelProperty">
    <xsd:sequence>
      <xsd:element name="content" type="FresnelContents" minOccurs="0"
      maxOccurs="1"/>
      <xsd:element name="label" type="FresnelLabel" minOccurs="1"
      maxOccurs="1"/>
      <xsd:element name="values" type="FresnelValues" minOccurs="1"
      maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="class" type="xsd:string"/>
    <xsd:attribute name="uri" type="xsd:string"/>
  </xsd:complexType>

  <xsd:complexType name="FresnelValues">
    <xsd:sequence>
      <xsd:element name="contents" type="FresnelContents" minOccurs="0"
```

```

        maxOccurs="1"/>
        <xsd:element name="value" type="FresnelValue" minOccurs="1"
            maxOccurs="unbounded"/>
    </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="FresnelValue">
    <xsd:choice>
        <xsd:element name="title" type="xsd:string" maxOccurs="1"/>
        <xsd:element name="resource" type="FresnelResource"
maxOccurs="1"/>
    </xsd:choice>
    <xsd:attribute name="class" type="xsd:string"/>
    <xsd:attribute name="output-type" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="FresnelLabel">
    <xsd:sequence>
        <xsd:element name="contents" type="FresnelContents" minOccurs="0"

            maxOccurs="1"/>
        <xsd:element name="title" type="xsd:string" minOccurs="1"
            maxOccurs="1"/>
    </xsd:sequence>
    <xsd:attribute name="class" type="xsd:string"/>
</xsd:complexType>

<xsd:complexType name="FresnelContents">
    <xsd:sequence>
        <xsd:element name="before" type="xsd:string" minOccurs="0"
            maxOccurs="1"/>
        <xsd:element name="after" type="xsd:string" minOccurs="0"
            maxOccurs="1"/>
        <xsd:element name="first" type="xsd:string" minOccurs="0"
            maxOccurs="1"/>
        <xsd:element name="last" type="xsd:string" minOccurs="0"
            maxOccurs="1"/>
    </xsd:sequence>
</xsd:complexType>
</xsd:schema>

```

A.1.2. SemVis Output Schema

SemVis can generate the XML output according to the following schema:

```

<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <xsd:annotation>
        <xsd:documentation xml:lang="en">
            Schema for SemVis output results.
        </xsd:documentation>
    </xsd:annotation>

    <xsd:complexType name="VisualizationElement">

        <xsd:attribute name="visible" type="xsd:boolean" />
        <xsd:attribute name="content" type="xsd:string" />
        <xsd:attribute name="fontSize" type="xsd:float" />
        <xsd:attribute name="color" type="xsd:string" />
        <xsd:attribute name="size" type="xsd:float" />

        <xsd:attribute name="width" type="xsd:float" />
        <xsd:attribute name="height" type="xsd:float" />
        <xsd:attribute name="depth" type="xsd:float" />
    </xsd:complexType>
</xsd:schema>

```

```

<xsd:attribute name="xPos" type="xsd:float" />
<xsd:attribute name="yPos" type="xsd:float" />
<xsd:attribute name="zPos" type="xsd:float" />

</xsd:complexType>

<xsd:complexType name="VisualizationBox"></xsd:complexType>

<xsd:element name="visualizationStructure"
  type="VisualizationStructure" />

<xsd:complexType name="VisualizationStructure">
  <xsd:sequence>
    <xsd:element name="resourceVisualization"
      type="ResourceVisualization" minOccurs="0" maxOccurs="unbounded" />
    <xsd:sequence minOccurs="0" maxOccurs="unbounded"
      type="scaleItem">
      <xsd:element name="scaleItem" type="ScaleItem" minOccurs="1"
        maxOccurs="unbounded" />
    </xsd:sequence>
  </xsd:sequence>
</xsd:complexType>

<xsd:complexType name="ResourceVisualization">
  <xsd:complexContent>
    <xsd:extension base="VisualizationElement">
      <xsd:sequence>
        <xsd:element name="staticContentVisualization"
          type="StaticContentVisualization" minOccurs="1" maxOccurs="1" />
        <xsd:element name="propertyVisualizationBox"
          type="PropertyVisualizationBox" minOccurs="0" maxOccurs="1" />
      </xsd:sequence>

      <xsd:attribute name="cycleDegree" type="xsd:float" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="StaticContentVisualization">
  <xsd:complexContent>
    <xsd:extension base="VisualizationElement">
      <xsd:sequence>
        <xsd:element name="label" type="LabelVisualization"
          minOccurs="1" maxOccurs="1" />
        <xsd:element name="description"
          type="MultilineTextVisualization" minOccurs="0" maxOccurs="1" />
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="PropertyVisualizationBox">
  <xsd:complexContent>
    <xsd:extension base="VisualizationBox">
      <xsd:sequence>
        <xsd:element name="propertyVisualization"
          type="PropertyVisualization" minOccurs="1" maxOccurs="unbounded"
        />
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="PropertyVisualization">

```

```

<xsd:complexContent>
  <xsd:extension base="VisualizationElement">
    <xsd:sequence>
      <xsd:element name="labelVisualization"
        type="LabelVisualization" minOccurs="1" maxOccurs="1"/>
      <xsd:element name="valueVisualizationBox"
        type="ValueVisualizationBox" minOccurs="1" maxOccurs="1"/>
    </xsd:sequence>
  </xsd:extension>
</xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="ValueVisualizationBox">
  <xsd:complexContent>
    <xsd:extension base="VisualizationBox">
      <xsd:sequence>
        <xsd:choice>
          <xsd:element name="valueVisualization"
            type="ValueVisualization" minOccurs="1" maxOccurs="unbounded" />
          <xsd:element name="resourceVisualization"
            type="ResourceVisualization" minOccurs="1" maxOccurs="unbounded"
          />
        </xsd:choice>
      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="ValueVisualization">
  <xsd:complexContent>
    <xsd:extension base="VisualizationElement">
      <xsd:sequence>

      </xsd:sequence>
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="LabelVisualization">
  <xsd:complexContent>
    <xsd:extension base="VisualizationElement">
      <xsd:attribute name="spacing" type="xsd:float" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="MultilineTextVisualization">
  <xsd:complexContent>
    <xsd:extension base="VisualizationBox">

    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

<xsd:complexType name="ScaleItem">
  <xsd:complexContent>
    <xsd:extension base="VisualizationElement">
      <xsd:attribute name="cycleDegree" type="xsd:float" />
    </xsd:extension>
  </xsd:complexContent>
</xsd:complexType>

</xsd:schema>

```